Orígenes y desarrollo de TOPS-20

por dan murphy

Copyright (c) 1989, 1996 - Dan Murphy

ORÍGENES

TOPS-20 se anunció por primera vez como un producto DEC y se envió en enero de 1976. El desarrollo comenzó en 1973 basado en TENEX [1], un sistema operativo para el PDP-10 que se usaba en varias instalaciones de investigación en todo el país. Muchos factores estuvieron involucrados en la decisión de comenzar el desarrollo de TOPS-20, pero principalmente, DEC quería un nuevo sistema operativo de memoria virtual para el procesador KL10 que se encontraba en las primeras etapas de desarrollo. Aunque se agregaron y cambiaron muchas cosas cuando TENEX se convirtió en TOPS-20 de 1973 a 1976, la mayor parte de su arquitectura se mantuvo tal como se diseñó en 1969 en Bolt Beranek _ & Newman Inc. (BBN) en Cambridge.

Por lo tanto, para comprender los factores que influyeron en el diseño de TOPS-20, debemos mirar hacia atrás a fines de la década de 1960 y al entorno en el que se creó TENEX. Algunas de las ideas que se incorporaron a TENEX, y el proceso que finalmente las llevó a DEC, tenían raíces que se remontan a principios de la década de 1960. En esta sección, rastrearemos estos aspectos de la historia de TOPS-20.

Fui estudiante universitario en el MIT de 1961 a 1965. Después de obtener mi título en 1965, comencé a trabajar para BBN realizando varias actividades de programación de sistemas y, en particular, apoyando el sistema LISP que entonces se usaba para la investigación de IA. Participé en el diseño de TENEX e implementé una serie de módulos de su kernel desde 1968 hasta 1972 y luego llegué a DEC junto con TENEX en 1973.

DEC compró TENEX en 1973, creyendo que representaba una buena base para un nuevo sistema comercial de última generación. Sin embargo, al igual que con otras "sensaciones de la noche a la mañana", este fue solo el paso final en un coqueteo entre DEC y BBN que se prolongó durante muchos años. En 1961, BBN fue uno de los primeros sitios externos a DEC en instalar y usar un PDP-1 (la primera computadora de DEC) e hizo algunos experimentos iniciales de tiempo compartido usando esta máquina y algunas modificaciones diseñadas localmente. Esto es generalmente reconocido como el primer sistema práctico de tiempo compartido. Durante los años siguientes, hubo muchas discusiones entre las dos empresas sobre la mejor manera de diseñar y construir computadoras.

Desarrollo de paginación en la década de 1960

Una de esas áreas de discusión fue la paginación y la gestión de la memoria. BBN estaba llevando a cabo una serie de proyectos de investigación de IA en LISP, y estos generalmente requerían una gran cantidad de memoria de acceso aleatorio. Pocas computadoras de ese día (y ninguna que BBN pudiera pagar) tenían el tamaño de memoria que se requería, por lo que se buscaron varias técnicas alternativas para solucionar la limitación [2]. Paginación y "memoria virtual" fue uno en el que hubo mucho interés. Había un sistema LISP ejecutándose en el PDP-1 en BBN, y lo había convertido para usar un sistema de paginación de software que había desarrollado anteriormente en el MIT.

Este sistema utilizaba un tambor de alta velocidad como almacenamiento de respaldo para la memoria principal y dividía tanto la memoria principal como el tambor en unidades de tamaño fijo (páginas) que podían copiarse de un lado a otro según fuera necesario. El sistema LISP en sí usaba punteros de 18 bits y se construyó como si hubiera una memoria de 18 bits (262 kpalabras) en la máquina. Sin embargo, con la paginación de software, cada referencia al objetivo de uno de estos punteros se cambió a una llamada de subrutina (o secuencia en línea). La subrutina tomaría los pocos bits de orden superior del puntero como un índice en una matriz (la tabla de páginas) que contendría la ubicación actual real de la página. Si la página estuviera entonces en la memoria principal, los bits del número de página de la tabla reemplazarían a los que se habían utilizado como índice en la tabla, y la referencia se completaría. Si la página deseada no estuviera en la memoria principal, se haría una llamada a una rutina de "administrador de página" para reorganizar las cosas según sea necesario. Eso significaba seleccionar una página en la memoria principal para moverla (copiarla) de regreso al tambor, y luego leer en esta página la página deseada del tambor. Finalmente, la tabla se ajustaría para reflejar estos cambios y la secuencia de referencia comenzaría de nuevo.

En el uso real, este sistema fue bastante efectivo. Los patrones de referencia de los típicos programas LISP que estábamos ejecutando eran tales que el sistema no estaba dominado por esperas de E/S de batería. Es decir, la mayoría de las referencias de punteros eran para páginas que estaban en la memoria principal y solo una pequeña fracción eran para páginas que tenían que leerse desde el disco*.

[* - Se han realizado varios estudios, algunos realizados por nosotros en BBN, para investigar el comportamiento de referencia de la memoria y recopilar estadísticas de varios programas [3].]

Sin embargo, el número real de referencias era lo suficientemente alto como para dedicar una gran cantidad de tiempo a la secuencia de traducción de direcciones de software, y nos dimos cuenta de que, en última instancia, esta traducción debe hacerse en hardware si se deseaba un sistema de memoria virtual paginado verdaderamente efectivo. construido.

Interés temprano en el PDP-6 y cancelación del primer producto de 36 bits.

El PDP-6 se anunció en 1964 y fue inmediatamente la máquina elegida para la programación LISP. Una sola palabra que contenía dos direcciones (CAR y CDR en términos de LISP) parecía diseñada especialmente para LISP (esto no era una mera coincidencia). El gran espacio de direcciones (para la época) de 2_^18 palabras ofrecía la promesa de una ejecución rápida y eficiente de grandes programas LISP. Aunque el conjunto de instrucciones básico no incluía una instrucción CONS (la primitiva LISP que crea listas), uno de los primeros PDP-6 tenía una modificación especial instalada para proporcionar esa operación. Esta fue la máquina que se instaló en el Laboratorio de IA de la Universidad de Stanford, dirigido por uno de los inventores de LISP, John McCarthy.

El grupo de BBN tenía toda la intención de adquirir un PDP-6 para mejorar nuestras capacidades LISP, pero, por supuesto, queríamos que tuviera paginación de hardware para poder transferir las técnicas de paginación de nuestro PDP-1 LISP. Se llevaron a cabo varias discusiones entre BBN y DEC, incluidos Win Hindle y Alan Kotok del lado de DEC. BBN estaba presionando para que la paginación se incluyera en un modelo PDP-6 posterior, o posiblemente como un complemento, pero

esta discusión llegó a su fin cuando DEC anunció que no habría más modelos PDP-6, es decir, que DEC estaba saliendo del negocio de los 36 bits. Esta (1966) fue la primera de varias ocasiones de este tipo.

Un desvío y una experiencia de aprendizaje

Tomando la palabra de DEC, BBN centró su atención en seleccionar otra máquina, con suerte una con un futuro en expansión, que podríamos usar para respaldar nuestros diversos proyectos de investigación. El resultado de este proceso de selección fue la compra de un SDS-940. SDS era Scientific Data Systems en El Segundo, California, luego adquirida por Xerox y conocida como XDS. El SDS-940 era una máquina de palabras de 24 bits con un espacio de direcciones virtuales modesto, pero tenía capacidades de paginación de hardware. Además, SDS estaba promocionando como una continuación, el Sigma-7 entonces en desarrollo, que sería más grande, más rápido y haría todo lo que podríamos desear. Si lo hizo, nunca lo supimos porque cuando salió, el DEC KA-10 había sido anunciado (o al menos filtrado), y felizmente volvimos a cambiar a la arquitectura que realmente queríamos.

También fue un factor que el Sigma-7 sufrió un pequeño retraso en el cronograma. Sí, esto realmente sucedió a pesar de las firmes garantías de SDS de que tal cosa no era posible. SDS incluso había publicado varios anuncios en revistas comerciales promocionando su próximo sistema como "el mejor sistema de tiempo compartido que aún no está disponible". Otro anuncio memorable decía "dijeron que debemos estar locos para publicar un cronograma firme de desarrollo de software, pero aquí está". Aparentemente, "ellos" estaban en lo correcto porque ese programa firme siguió el camino de la mayoría de los programas de desarrollo de software "firmes" de la época.

El anuncio anterior también inspiró a alguien en DEC a crear un anuncio de contador para TOPS-10 que decía "anunciando el mejor sistema de tiempo compartido disponible".

Sin embargo, BBN usó el SDS-940 durante un par de años y ejecutamos un sistema operativo desarrollado por un grupo en U.C. Berkeley. Eso fue significativo porque una serie de características en el sistema de tiempo compartido de Berkeley se modificaron y adoptaron posteriormente en TENEX.

La máquina de 36 bits renace.

Cuando BBN se enteró de que DEC había regresado en secreto al negocio de los 36 bits (o al menos estaba desarrollando una nueva máquina con esa arquitectura), presionamos una vez más para que se incluyera un sistema de paginación de hardware. Y una vez más, esto fue en vano. Un avance en el KA-10 fue la protección dual y los registros de reubicación (el PDP-6 tenía solo un par). Esto permitió que los programas se dividieran en dos segmentos: una porción reentrante de solo lectura y un área de datos de lectura y escritura. Eso, sin embargo, era lo más lejos que DEC estaba dispuesto a llegar en ese momento para avanzar en el soporte de gestión de memoria del sistema operativo de última generación.

Otro factor fue la firme intención de DEC de mantener bajo el precio de esta nueva máquina grande.

A DEC le estaba yendo bien con máquinas pequeñas (en particular, PDP-5 y PDP-8), pero el PDP-6 había sufrido numerosos problemas de ingeniería y confiabilidad, era difícil de vender y, según algunos, casi arruina la empresa. El KA-10 tenía una restricción de diseño que permitía configurar y vender una máquina mínima por menos de 100 000 dólares. Esta configuración estaba limitada a 16 kpalabras de memoria principal y no tenía registros de estado sólido para los 16 acumuladores (todas las referencias de CA iban a la memoria principal con un costo considerable en velocidad). Para respaldar esto, se diseñó una compilación "pequeña" de TOPS-10 que cabría en 8 kwords, y se exprimieron varias utilidades de manera similar: el PIP de 1K, etc. Es difícil decir si el precio de entrada de $ 99,999 fue una buena estrategia de marketing. En cualquier caso, nunca se vendió ninguno de esa configuración.

Además, la paginación y la "memoria virtual" todavía eran conceptos bastante nuevos en ese momento. Los sistemas comerciales significativos aún tenían que adoptar estas técnicas, y la idea de pretender tener más memoria de la que realmente tenía fue vista con mucho escepticismo en muchos sectores.

El requisito de paginación conduce a un nuevo sistema operativo

Sin desanimarse por la negativa de DEC a ver la sabiduría de nuestro enfoque, BBN planeó la compra de varios KA10 y se dispuso a descubrir cómo convertirlos en el sistema que realmente queríamos. Además de la arquitectura de 36 bits que preferíamos, el sistema KA10 había conservado gran parte de la modularidad del hardware del PDP-6, incluido en particular el bus de memoria con unidades de memoria independientes. Esto nos llevó a concebir la idea de colocar un dispositivo de mapeo entre el procesador y las memorias que realizaría la traducción de direcciones como queríamos.

Otra decisión que tuvimos que tomar fue si intentar o no modificar TOPS-10 para admitir la paginación bajo demanda. La alternativa era construir un nuevo sistema desde cero, una empresa ambiciosa incluso en aquellos días. Obviamente, decidimos construir un nuevo sistema, el sistema que más tarde se llamó TENEX. Esta decisión se justificó en gran medida porque se requeriría una cirugía mayor para adoptar TOPS-10 como deseábamos, e incluso entonces probablemente no podríamos resolver todos los problemas. En retrospectiva, esta opinión probablemente estaba justificada ya que, aunque el desarrollo de TOPS-10 continuó durante casi 20 años después de que comenzamos con TENEX, TOPS-10 nunca se modificó para tener todas las funciones de memoria virtual de TENEX/TOPS-20. TOPS-10 finalmente admitió la paginación y la memoria virtual, pero no las otras características.

Más allá de eso, había una serie de otras funciones no relacionadas con la paginación que queríamos en un sistema operativo. Esto inclinó aún más la decisión hacia la implementación de un nuevo sistema. Por otro lado, no teníamos el deseo o la capacidad de implementar nuevas versiones de los muchos compiladores y utilidades de lenguaje disponibles en TOPS-10, por lo que se necesitaba un método para ejecutar estas imágenes "tal cual". Decidimos que sería posible emular las llamadas del sistema operativo de TOPS-10 con un módulo separado que traduciría las solicitudes en servicios nativos equivalentes. Esto se describe con más detalle más adelante. Entonces, el plan sería implementar un nuevo sistema desde el nivel de interfaz del sistema operativo hacia abajo, además de sistemas principales particulares como LISP que fueron clave para nuestro trabajo de

investigación.

La memoria virtual mediante paginación fue la necesidad fundamental que llevó a la decisión de construir TENEX. Si TOPS-10 u otro sistema hubiera estado disponible que solo admitiera ese requisito, probablemente lo habríamos utilizado y no emprendido la construcción de un nuevo kernel. Los otros deseos que teníamos para nuestro sistema no habrían sido suficientes para justificar tal esfuerzo y, por supuesto, no previmos todas las capacidades que eventualmente tendría.

ORIGEN DE LAS PRINCIPALES CARACTERÍSTICAS

El desarrollo de TENEX estuvo motivado en gran medida por las necesidades de varios programas de investigación en BBN, en particular la Inteligencia Artificial. Estas necesidades fueron compartidas por varios otros laboratorios apoyados por la Agencia de Proyectos de Investigación Avanzada (ARPA) del DOD, y en el período de 1970 a 1972, varios de ellos instalaron TENEX y lo convirtieron en la base de sus programas de investigación. Como se señaló anteriormente, una gran memoria virtual era la principal de estas necesidades debido a la popularidad y el tamaño de los programas LISP para reconocimiento de voz, procesamiento de imágenes, procesamiento de lenguaje natural y otros proyectos de IA.

Durante este mismo período, la red ARPA se estaba desarrollando y entró en funcionamiento. TENEX fue uno de los primeros sistemas en conectarse a ARPANET y en tener soporte de sistema operativo para la red como una capacidad general del sistema. Esto aumentó aún más su popularidad.

Como la mayoría de los otros sistemas de software, TENEX fue una amalgama de ideas de varias fuentes. La mayoría de las características que llegaron a ser apreciadas en TOPS-20 tenían raíces en otros sistemas. Algunos se tomaron prácticamente sin cambios, algunos se modificaron de manera que resultaron ser críticos y otros sirvieron simplemente como el germen de una idea difícilmente reconocible en el sistema final. Entre los que podemos rastrear a uno o más sistemas anteriores están: "reconocimiento de escape", estructura de memoria virtual, estructura de procesos y sus usos, y técnicas de programación de procesos de tiempo compartido.

Tres sistemas afectaron más directamente el diseño de TENEX: el sistema MULTICS en el MIT, el sistema DEC TOPS-10 y el sistema de tiempo compartido de Berkeley para la computadora SDS 940. MULTICS era el sistema más nuevo y "vanguardista" de la época, e incorporaba las últimas ideas en la estructura del sistema operativo. De hecho, era popular en algunos círculos decir que, con la implementación de MULTICS, "el problema del sistema operativo había sido resuelto". Varios miembros de la facultad y el personal del MIT que habían trabajado en MULTICS brindaron una valiosa revisión y comentarios sobre el diseño emergente de TENEX.

Muchos de los conceptos de paginación provienen de nuestro propio trabajo anterior, el sistema PDP-1 LISP en particular. Otras ideas habían aparecido recientemente en la literatura del sistema operativo, incluido el modelo de "conjunto de trabajo" del comportamiento del programa de Peter J. Denning [4]. MULTICS había desarrollado el concepto de segmentación y de mapeo de procesos de archivo, es decir, usar el espacio de direcciones virtuales como una "ventana" en los datos que residen permanentemente en un archivo.

El sistema Berkeley 940 tenía una estructura multiproceso, con procesos que eran relativamente económicos de crear. Eso, a su vez, permitió cosas como un intérprete de lenguaje de comandos del sistema que se ejecutaba en modo sin privilegios y un depurador que no compartía el espacio de direcciones con el programa bajo prueba y, por lo tanto, no estaba sujeto a ser destruido por un programa fuera de control.

Memoria virtual y espacio de direcciones de proceso

Estos dos conceptos, la memoria virtual y los procesos múltiples, fueron fundamentales para el diseño de TENEX y, en última instancia, fueron clave para el poder y la flexibilidad de TOPS-20. Las dos abstracciones también funcionaron bien juntas. El concepto de un proceso incluía un espacio de direcciones virtuales de 262 kpalabras (direcciones de 18 bits), el máximo posible bajo el diseño del conjunto de instrucciones de 36 bits. Todavía faltaban muchos años para el direccionamiento extendido, y la posibilidad de que alguna vez se necesitaran más de 262k rara vez se le ocurrió a nadie.

Simplemente queríamos que el espacio completo de direcciones virtuales estuviera disponible para todos los procesos, sin necesidad de que el programa en sí estuviera al tanto de la paginación de demanda o de "volverse virtual". Creíamos que los mecanismos podrían integrarse en el hardware de paginación y el sistema operativo para rastrear y determinar los conjuntos de trabajo del proceso, y haría un trabajo tan bueno en la administración de la memoria de manera eficiente como si el programa proporcionara información explícitamente (lo que podría ser incorrecto). Por supuesto, podría suceder que un programa en particular tuviera un conjunto de trabajo demasiado grande para ejecutarse de manera eficiente en la memoria física disponible, pero el sistema aún haría lo mejor que pudiera y el programa progresaría a costa de una considerable paliza del tambor. . En cualquier caso, la implementación de la memoria virtual debía ser transparente para los programas de usuario.

Información de historial de referencia de página

Para admitir una memoria virtual paginada por demanda transparente, habíamos aprendido que teníamos que mantener información histórica sobre el comportamiento de los programas y el uso de las páginas. Esto condujo al diseño y la inclusión de la "tabla de estado del núcleo" en la interfaz de paginación del hardware. Otras tablas (es decir, tablas de páginas) contendrían información que proporcionaría la traducción de direcciones virtuales a físicas; la tabla de estado central se especificó para contener información relacionada con el uso dinámico y el movimiento de páginas. Para cada página física en uso para la paginación, la tabla de estado central registraría: (1) una marca de tiempo de la última referencia y (2) una identificación de los procesos que habían hecho referencia a la página. Este último se incluyó porque esperábamos que a menudo hubiera muchos procesos compartiendo el uso de una página determinada. La necesidad de compartir también se representó en varios otros aspectos del diseño de paginación[5].

Páginas compartidas y asignación de archivos a procesos

La necesidad percibida de compartir páginas de memoria y las ideas de segmentación que habíamos visto en Multics llevaron al diseño de las capacidades de mapeo de archivo a proceso. En el

hardware de paginación, esto estaba respaldado por dos tipos de entradas de tablas de páginas, "compartidas" e "indirectas", además del tipo "privado" que contendría un mapeo directo de virtual a físico. Esto permitió que la dirección física de una página compartida se mantuviera en un solo lugar aunque la página se usara en muchos lugares, incluso en el espacio de direcciones de los procesos que se intercambiaron. El puntero "indirecto" eventualmente permitió otra capacidad que inicialmente no previmos: la capacidad de hacer que dos o más espacios de direcciones de proceso sean equivalentes, no solo con respecto a los datos contenidos en ellos, sino también con respecto al mapeo*.

[* - Esto permite, por ejemplo, una operación de "bifurcación" de tipo Unix para crear el espacio de direcciones secundario de forma muy económica, ya sea que se reemplace o no posteriormente por un EXEC.]

En el software, queríamos lograr algunos de los beneficios de la "segmentación" de Multics, pero el espacio de direcciones del PDP-10 no era lo suficientemente grande para usar el enfoque de Multics directamente. En consecuencia, decidimos extender la idea de la memoria con nombre hasta el nivel de página y permitir que el mapeo se realice página por página. El resultado de esto fue el diseño fundamental de la arquitectura de memoria virtual TENEX/TOPS-20, específicamente:

Un espacio de direcciones de proceso no contiene almacenamiento real, es simplemente un conjunto de 512 ventanas (asignaciones) para el almacenamiento.
El único almacenamiento real reside en los archivos. Una página de almacenamiento real se identifica por la combinación de: nombre de archivo + VPN (número de página virtual).
Una llamada al sistema asigna páginas de almacenamiento real a ventanas de proceso y las desasigna. La misma página se puede asignar a muchos procesos (y a varias ventanas del mismo proceso), y todas las ventanas ven y comparten el mismo almacenamiento físico. Copiar en páginas de escritura

Sabíamos que la mayor parte del intercambio sería de datos de solo lectura, especialmente de texto de programa y datos constantes, pero que cualquier programa podría contener variables inicializadas que eventualmente se escribirían. Más allá de eso, casi todos los datos que normalmente son de solo lectura pueden modificarse en algunas circunstancias, p. inserción de un punto de interrupción para la depuración. En el sistema LISP, grandes cantidades de programas se representaban como estructuras LISP, y el usuario podía editar y modificar estas estructuras.

Por lo tanto, no era aceptable imponer referencias de solo lectura en la mayor parte de este almacenamiento, ni sería eficiente copiarlo cada vez que un proceso lo usara o hiciera referencia a él. Eso anularía gran parte del objetivo de compartir. La solución que ideamos fue la página de "copia sobre escritura". Esto significaba que la página permanecería compartida siempre que solo se le hicieran referencias de lectura o ejecución, pero si un proceso intentaba escribirla, se haría una copia y el mapa del proceso se ajustaría para contener la copia. Esto ocurriría de forma transparente al proceso. La aplicación real de solo lectura estaba disponible, por supuesto, y el contenido real del archivo (por ejemplo, de un archivo ejecutable del programa del sistema) estaba adecuadamente protegido contra modificaciones, pero el usuario podía asignar cualquiera de esas páginas de copia en escritura, y una referencia de escritura sería entonces simplemente resultará en el uso de una copia modificada.

Lecciones de diseño aprendidas

Se puede decir que Multics ha contribuido con algo más que las ideas para la organización de la memoria virtual y otras capacidades específicas. Durante el diseño de TENEX, invitamos a algunos de los diseñadores e implementadores de Multics para revisar nuestro progreso y decisiones. Como suele ser el caso, habíamos caído en la trampa de tratar de hacer demasiado en varias áreas y habíamos producido algunos diseños que eran bastante intrincados y complejos. Varias personas de Multics nos golpearon en esas ocasiones, diciendo "esto es demasiado complicado, ¡simplifique! ¡Tira esto! ¡Deshazte de eso!" Seguimos gran parte de ese consejo (y probablemente podríamos haber tomado más), y doy crédito a estas revisiones por hacer que las capacidades básicas sean significativamente más fáciles de entender y programar, así como de implementar y depurar.

Esta fue una de las primeras ocasiones en las que realmente comencé a apreciar el hecho de que un diseño complejo y complicado para alguna instalación en un producto de software o hardware generalmente no es una indicación de gran habilidad y madurez por parte del diseñador. Más bien, es típicamente evidencia de falta de madurez, falta de perspicacia, falta de comprensión de los costos de la complejidad y falta de visión del problema en su contexto más amplio.

El tiempo que transcurrió desde que se escribió la primera palabra del código TENEX hasta que el sistema estuvo activo y sirviendo a los usuarios regulares fue de aproximadamente seis meses, algo notable incluso para esa época. El sistema estaba lejos de estar completo en ese momento, pero los usuarios internos de BBN estaban haciendo un uso efectivo de él.

Una de las cosas que contribuyó a un desarrollo tan rápido fue nuestra preocupación por el soporte de depuración. DDT era el depurador simbólico estándar en el PDP-10 y se había desarrollado a partir de versiones anteriores en el PDP-6, PDP-1 y TX-0 (una máquina experimental en el MIT que precedió al PDP-1). DDT tenía mucho poder para su época, e incluso hoy en día sigue siendo la elección de algunos usuarios experimentados a pesar de la disponibilidad de sistemas de depuración orientados a pantallas y ventanas. DDT se ejecutó tanto en modo ejecutivo (núcleo) como en modo de usuario bajo TOPS-10, y también admitimos ambos en TENEX.

Además, proporcionamos una forma de que DDT se ejecute en tiempo compartido mientras observa el código del sistema y las estructuras de datos. Esto permitió que varios desarrolladores estuvieran depurando el código del sistema al mismo tiempo, al menos hasta cierto punto. Incluso los puntos de interrupción podrían manejarse, siempre que ningún otro proceso ejecutara el código mientras el punto de interrupción estaba en su lugar. En todos los modos, DDT tenía el conjunto completo de símbolos disponibles, por lo que pudimos depurar sin una referencia constante a los listados y la escritura de valores numéricos.

En el modo de usuario, se puede agregar DDT a cualquier programa incluso si ya se está ejecutando. Los sistemas anteriores requerían que la depuración se anticipara vinculando el depurador con el programa. En TENEX, DDT simplemente se mapeaba cuando era necesario en una dirección alta no utilizada en el espacio de direcciones bien fuera del camino de la mayoría de los programas. Una versión aún más poderosa de DDT, llamada IDDT (por invisible) se ejecutó en un proceso separado del programa que se está depurando. Por lo tanto, era inmune a la corrupción

de la memoria por parte del programa bajo prueba, y podía interceptar todas y cada una de las acciones ilegales. Desafortunadamente, esta versión de DDT divergió de la corriente principal DEC DDT, e IDDT nunca se actualizó y se convirtió en el depurador normal, incluso bajo TOPS-20. No obstante, el uso fácil y extenso de DDT seguramente hizo que el desarrollo fuera más efectivo, tanto para los usuarios como para los programadores de sistemas.

Otra área en la que TENEX, y más tarde TOPS-20, se destacaron fue en la interfaz de usuario, que se describe en detalle a continuación. Una razón clave por la que pudimos ampliar la interfaz de usuario de manera tan significativa es que el código que la implementó se ejecutó en un programa de usuario normal. Por lo tanto, no estaba limitado por el tamaño ni por funciones que no se podían realizar en modo kernel. Más allá de eso, la depuración fue mucho más simple: no requirió el uso independiente de la máquina como lo hizo la depuración del kernel. Todo esto permitió una gran cantidad de experimentación y evolución basada en el ajuste y la retroalimentación del uso real y los usuarios reales.

Filosofía de diseño orientada al usuario

Una parte de la "filosofía" de diseño de sistemas había surgido en BBN y entre algunos de los sitios de investigación de ARPA que iba a tener un gran impacto en la sensación general de TENEX y, en última instancia, TOPS-20. En ese momento, lo llamamos "ingeniería humana": queríamos que el sistema fuera fácil de aprender y de usar, y queríamos que el sistema se ocupara de la mayor cantidad posible de detalles sucios para que el programador o el usuario no hay que lidiar con ellos. Más allá de eso, estábamos dispuestos a gastar ciclos de máquina reales y memoria real (o al menos virtual) para que esto sucediera.

Esta filosofía condujo inicialmente a las funciones de interfaz humana de EXEC, incluido el "reconocimiento de escape", la función de ayuda con el signo de interrogación, los subcomandos opcionales y las palabras de "ruido". Pocas personas ahora argumentan en contra de la necesidad de proporcionar interfaces humanas efectivas, pero en ese momento hubo muchos detractores que sintieron que era una pérdida de ciclos hacer cosas como el reconocimiento de comandos. Este tipo de cosas, dijeron, "ralentizarían el sistema" y evitarían que se hiciera un "trabajo útil". Otros sistemas contemporáneos usaban comandos cortos, a menudo de una letra, y argumentos de comando, no proporcionaban ayuda en línea y no daban ninguna respuesta al usuario aparte de la "respuesta" (si la hubiera) al comando que se había ingresado.

Uso de la computadora por no expertos

Muchos de estos sistemas cayeron en la categoría "por expertos, para expertos". Es decir, fueron construidos por expertos y destinados a ser utilizados por otros expertos. Obviamente, un "experto" no necesitaría funciones frívolas como palabras irrelevantes o reconocimiento de comandos. Los expertos sabrían, no solo todos los comandos, sino también todas las abreviaturas legales. Los expertos pueden haber leído el manual una vez, pero siempre recordarán todo lo necesario para interactuar con el sistema. Así iban las suposiciones implícitas sobre los usuarios: o eras un experto, o eras un ser inferior que no tenía realmente derecho a usar la computadora de todos modos.

El equipo de TENEX tuvo una opinión diferente. Estaba claro que el poder de las computadoras

aumentaba cada año, por lo que uno debería esperar que la computadora hiciera más y más cosas interesantes. La mayoría de la gente estaría de acuerdo en que uno de los propósitos principales de la automatización informática es liberar a las personas de tareas aburridas y repetitivas; creíamos que ese propósito también se extendía al desarrollo de sistemas informáticos, y que era incorrecto exigir que una persona aprendiera un nuevo conjunto de tareas arcanas y aburridas para poder usar la computadora. La máquina debe adaptarse al hombre, no el hombre a la máquina.

La opinión probablemente se vio reforzada por la investigación de inteligencia artificial que se está realizando en el entorno donde se diseñó TENEX. En áreas tales como el reconocimiento de voz, el reconocimiento de patrones y la comprensión del lenguaje natural, se estaba aplicando un poder de cómputo masivo para hacer que las computadoras interactúen de maneras que serían más humanas. Estos fueron esfuerzos a largo plazo, pero también queríamos que nuestros sistemas informáticos estuvieran más orientados a los humanos en el término medio.

Una de las ideas que surgieron de la investigación de la IA fue que la computadora debería "hacer lo que quiero decir", incluso si no lo dije del todo bien o tal vez cometí un error de ortografía. Warren Teitelman implementó amplias capacidades en LISP para corregir errores automáticamente o con una intervención mínima del usuario, y esto se denominó paquete DWIM. Con este tipo de esfuerzo de investigación en curso, éramos sensibles a las restricciones y torpezas de la mayoría de los sistemas existentes.

Por lo tanto, tomamos decisiones (ahora obvias) como (1) identificar a los usuarios por nombre alfanumérico en lugar de número; (2) permitir que los nombres de los archivos sean lo suficientemente largos para representar información significativa (para el ser humano); (3) escribir o mostrar una confirmación positiva de las acciones solicitadas por el usuario; (4) permitir una variedad de preferencias individuales entre los usuarios con respecto al estilo de entrada y salida de comandos.

Interacciones juzgadas sobre una base humano-humano

Buscamos específicamente formas de interacción que parecieran naturales cuando se juzgan desde una perspectiva de persona a persona. Sobre esa base, muchos comportamientos típicos de la interfaz de la computadora son bastante hostiles. Respuestas concisas, crípticas y/o abruptas; sintaxis de entrada inflexible y oscura; uso de palabras para significados que son contrarios a la intuición: todas estas cosas se juzgan como groseras o peores si se ven en una interacción de persona a persona.

Este deseo de hacer que la computadora hiciera el trabajo grunge también motivó el diseño en otras áreas. La paginación bajo demanda y la memoria virtual se consideraron formas de reducir la carga y la dificultad de escribir programas grandes. Con una gran memoria virtual, el programador no tendría que preocuparse por segmentar y superponer su programa. Más bien, podría dejar la gestión de la memoria al sistema y concentrarse en los detalles de la tarea en cuestión. Otras funciones, como la estructura del proceso, las llamadas a las utilidades del sistema y el análisis de archivos estandarizado, tenían como objetivo facilitar el trabajo de programación del sistema al proporcionar funciones estándar y potentes al programador.

Reconocimiento de escape, palabras ruidosas, ayuda

Una de las características más favorecidas entre los usuarios de TOPS-20, y una de las más identificadas con el propio TOPS-20, es el "reconocimiento de escape". Con esto, el usuario a menudo puede hacer que el sistema, en efecto, escriba la mayor parte de un comando o nombre simbólico. La función se usa más fácilmente de lo que se describe; sin embargo, a continuación se presenta una breve descripción para ayudar a comprender su desarrollo.

Una breve descripción del reconocimiento y la ayuda

Escribir la tecla de escape le dice al sistema: "si sabe lo que quiero decir con lo que he escrito hasta este punto, escriba lo que viene a continuación como si lo hubiera escrito". Lo que se muestra en la pantalla o mecanografiado parece como si el usuario lo hubiera escrito, pero por supuesto, el sistema lo escribe mucho más rápido. Por ejemplo, si el usuario escribe DIR y escapa, el sistema continuará la línea para que se lea DIRECTORIO.

TOPS-20 también acepta solo la abreviatura DIR (sin escape), y el usuario experto que quiera ingresar el comando en forma abreviada puede hacerlo sin demora. Para el usuario novato, escribir escape tiene varios propósitos:

Confirma que la entrada ingresada hasta ese momento es legal. Por el contrario, si el usuario cometió un error, se entera de inmediato en lugar de invertir el esfuerzo adicional y, en última instancia, desperdiciado para escribir el resto del comando.
Confirma para el usuario que lo que el sistema entiende ahora es (o no es) lo que el usuario quiere decir. Por ejemplo, si el usuario escribe DEL, el sistema completa la palabra DELETE. Si el usuario hubiera estado pensando en un comando DEMORA, sabría inmediatamente que el sistema no había entendido lo que quería decir.
Escribir escape también hace que el sistema responda con cualquier palabra de "ruido" que pueda ser parte del comando. Una palabra irrelevante no es sintáctica ni semánticamente necesaria para el comando, pero sirve para que sea más legible para el usuario y para sugerir lo que sigue. Escribir DIR y escapar en realidad hace que la pantalla muestre:

DIRECTORIO (DE ARCHIVO)

Esto le indica al usuario que los archivos se están tratando en este comando y que se puede dar un archivo como la siguiente entrada. En un comando con varios parámetros, este tipo de interacción puede ocurrir varias veces. Se ha demostrado claramente en este y otros entornos que la interacción frecuente y la retroalimentación como esta son de gran beneficio para darle al usuario la confianza de que va por el camino correcto y que la computadora no está esperando para lanzar una trampa terrible si dice Ocurre algo. Si bien puede llevar un poco más de tiempo ingresar un comando de esta manera que si lo ingresara un experto usando las abreviaturas más cortas, ese costo es pequeño en comparación con la penalización de ingresar un comando incorrecto. Un comando incorrecto significa al menos que se ha desperdiciado el tiempo dedicado a escribir la línea de comando. Si da como resultado que se tome alguna acción errónea (en lugar de no tomar ninguna acción), el costo puede ser mucho mayor.

Esta es una razón subyacente clave por la que la interfaz TOPS-20 se percibe como amigable: reduce significativamente la cantidad de grandes eventos de retroalimentación negativa que le ocurren al usuario y, en cambio, proporciona muchas más interacciones pequeñas pero positivas (es decir, exitosas). Este refuerzo positivo se consideraría bastante obvio si se ve en términos de interacción de persona a persona, pero a lo largo de la mayor parte de la historia de las computadoras, hemos ignorado la necesidad del usuario humano de que la computadora sea un miembro positivo y alentador del diálogo. .

Escribir escape es solo una solicitud. Si su entrada hasta el momento es ambigua, el sistema simplemente emite una señal (con una campana o un pitido) y espera de nuevo por más entradas. Además, el reconocimiento de escape está disponible para nombres simbólicos (por ejemplo, archivos), así como para verbos de comando. Esto significa que un usuario puede usar nombres de archivo largos y descriptivos para ayudar a realizar un seguimiento de lo que contienen los archivos, sin tener que escribir estos nombres largos en cada referencia. Por ejemplo, si mi directorio contiene:

BIG_PROGRAM_FILE_SOURCE
TEXTO_MANUAL_MUY_LARGO

Solo necesito escribir B o V para identificar sin ambigüedades uno de esos archivos. Escribir letras extra antes de la fuga no duele, así que no tengo que pensar en la abreviatura mínima; Puedo escribir VER y ver si el sistema reconoce el archivo.

Ayuda de signo de interrogación

Finalmente, si el usuario aún no está seguro de qué entrada viene a continuación en un comando, escribe un signo de interrogación y el sistema proporcionará una lista de opciones que son legales en ese momento. La lista incluye palabras clave específicas (p. ej., ARCHIVO, DIRECTORIO) y descripciones genéricas (p. ej., "archivo de entrada"). Lo que es más importante, la solicitud del signo de interrogación no destruye la entrada del comando anterior. Después de que se muestra la lista de alternativas, se vuelve a mostrar el comando parcial y la entrada continúa desde el punto justo antes de que el usuario haya escrito el signo de interrogación.

Como resultado de esta función:

Los usuarios nunca tienen que ir a buscar un manual y buscar tratando de encontrar el nombre de una palabra reservada olvidada (comando o parámetro). Esto elimina el aspecto "Sé una palabra, ¿puedes adivinarla?" de muchas interfaces de computadora.
El usuario a menudo puede averiguar a partir de la elección de parámetros lo que hará un comando u opción desconocido. Esto elimina aún más la laboriosa búsqueda de manuales.
Debido a que el contexto del comando actual no se pierde cuando se solicita ayuda, el usuario puede ir paso a paso a través de un comando, descifrando cada campo por turno. En los sistemas en los que obtener ayuda es un comando en sí mismo, es posible que el usuario deba escribir un comando largo y desconocido en una hoja de papel para poder ingresarlo por completo.

A medida que las interfaces orientadas a menús se han vuelto más utilizadas, la ventaja de tener todas las opciones visibles para el usuario se ha vuelto obvia. La función de ayuda con el signo de interrogación puede, en retrospectiva, verse como un tipo de enfoque de "menú a pedido", y funcionó incluso en terminales demasiado lentos para admitir interfaces completas basadas en menús.

Origen del reconocimiento

El sistema de tiempo compartido de Berkeley para el SDS-940 tenía una forma anterior de reconocimiento. Sin embargo, no usó escape. En ese sistema, el reconocimiento era automático; cada vez que el usuario había escrito lo suficiente para identificar sin ambigüedades un comando o un nombre simbólico, el sistema cobraba vida y escribía el resto. Esto hizo que las pulsaciones de teclas fueran mínimas en el caso ideal, pero tenía un problema importante y, en última instancia, fatal: si el usuario escribía demasiado en un campo, es decir, más de la cantidad necesaria para que el sistema reconozca ese campo, la entrada sería en el siguiente campo donde no estaba previsto. Por ejemplo, el sistema reconocería COP como suficiente para COPY y proporcionaría la "Y". Pero si escribieras el verbo completo, obtendrías:

```
* COPIAR Y
|
|- escrito por la computadora
```

Entonces, al menos tendrías que borrar la "Y" adicional. Si no se dio cuenta de lo que sucedió y continuó escribiendo el resto del comando, lo que pretendía era:

```
* COPIAR OLDFIL NEWFIL
```

saldría como

```
* COPIAR Y OLDFIL NEWFIL
```

Esto al menos produciría un error, y en casos patológicos podría causar daños importantes. [En el ejemplo anterior, tenga en cuenta que el nombre de archivo anterior termina en el campo de parámetro de archivo nuevo.]

Con la experiencia, los usuarios llegarían a conocer las abreviaturas de las palabras reservadas y, por lo tanto, rara vez caerían en la trampa de este ejemplo. Sin embargo, este reconocimiento automático también funcionó en archivos en directorios de archivos, y allí, la abreviatura mínima podía cambiar de minuto a minuto a medida que se agregaban y eliminaban archivos. Por lo tanto, incluso un usuario experimentado tendría que escribir con cuidado. Curiosamente, este sistema favorecía a los mecanógrafos lentos que se beneficiaban de tener que escribir la cantidad mínima de

entrada, pero era una fuente de frustración enloquecedora para los mecanógrafos rápidos que a menudo superaban el sistema. Y cuando el sistema se cargó y la respuesta disminuyó, fue un desastre para todos.

Esta experiencia nos convenció de que no queríamos perpetuar el reconocimiento automático de esa forma en TENEX. Pasamos mucho tiempo pensando en formas de resolver el problema de "sobrescribir", incluidas reglas como "si hay una entrada adicional y coincide con la última palabra clave después del punto de reconocimiento, aplíquela a eso", etc., pero finalmente no pudo encontrar ningún método que no dejara abierta una gran clase de errores dependientes del tiempo. Consideramos un modo que habilitaría o deshabilitaría el reconocimiento automático, pero decidimos que no haría que la función fuera más utilizable.

Por lo tanto, decidimos hacer que el reconocimiento sucediera solo a pedido explícito, aunque se requeriría una pulsación de tecla adicional. Seleccionamos la tecla de escape para este propósito.

Evitar modos

En general, se convirtió en nuestro objetivo evitar los modos. Creíamos que definir los modos "experto", "principiante", etc. sería ignorar el hecho de que una persona puede ser novata y experta al mismo tiempo, y puede tener preferencias que no se ajustan a ninguna de las definiciones. Es probable que incluso un usuario experimentado no esté familiarizado (y por lo tanto sea un "novato") con algunas partes del sistema, algunos comandos, etc., y desearía recibir indicaciones adicionales cuando trabaje en esas áreas. Con el tiempo, observamos que algunos usuarios muy experimentados continúan utilizando las indicaciones de manera extensiva debido a la retroalimentación positiva que proporciona.

TENEX SE MUEVA A DIC

En retrospectiva, las circunstancias bajo las cuales TENEX se mudó a DEC y se convirtió en TOPS20 parecen haber incluido una serie de eventos fortuitos. Como se señaló anteriormente, en 1972, TENEX había alcanzado cierta popularidad entre los investigadores de la Red ARPA. La renacida máquina de 36 bits, recién bautizada como DECsystem-10, también disfrutaba de un éxito razonable en varios otros mercados. Sin embargo, valía la pena publicitar el prestigio de ser la elección de los investigadores de vanguardia, por lo que DEC publicó un anuncio en varias publicaciones comerciales titulado "ARPA tiene una red de supercomputadoras" y señaló que una gran fracción de ellas eran DECsystem-10. De hecho, la mayoría de ellos estaban ejecutando TENEX. Para abril de 1972, había siete sitios además de BBN que ejecutaban TENEX.

BBN tenía un negocio modesto construyendo el hardware de paginación externo que, con la tecnología de ese día, requería un gabinete de lógica alto de 19 pulgadas de ancho. Mientras tanto, DEC había comenzado a trabajar en una máquina sucesora que se conocería como KI10 (la "I" al menos sugiere la tecnología de circuito integrado que se iba a utilizar). Ya en junio de 1970, se llevaron a cabo reuniones en las que la gente de BBN intentó persuadir a DEC para que incluyera hardware de buscapersonas similar al diseño del buscapersonas de BBN. Eventualmente, DEC decidió incluir paginación en el KI10, pero estaba basado en una arquitectura mucho más simple. Los ingenieros de DEC no estaban convencidos de que las distintas formas de punteros (privados,

compartidos, indirectos) y la tabla de estado del núcleo valdrían la cantidad de hardware necesaria para soportarlos. No obstante, eligieron el mismo tamaño de página, 512 palabras, lo que al menos dejó abierta la puerta a algún tipo de acomodación posterior.

KI-TENEX

Cuando salió el KI10, DEC estaba decepcionado (por decir lo menos) por la falta de interés entre la comunidad de investigación que había ayudado a impulsar las ventas del KA10. El problema era que la máquina no funcionaba con TENEX. Era casi el doble de rápido que el KA10, pero la paginación era diferente de lo que requería TENEX. También fue bastante evidente que no sería práctico modificar la estructura de TENEX para utilizar la paginación KI10 más limitada. Como ironía adicional, la versión de TOPS-10 enviada inicialmente con el KI10 usaba el hardware de paginación solo para simular el hardware de protección y reubicación del KA10 y no obtuvo ningún beneficio de él.

Durante el verano de 1972, había decidido buscar nuevas oportunidades fuera de BBN. Como era de esperar, una empresa con la que hablé fue DEC. En el curso de esas discusiones, me preguntaron sobre la posibilidad de poner TENEX en el KI10. Este no era un deseo generalizado dentro de DEC en general o dentro de la ingeniería de software de DEC en particular, pero era de gran interés para Allan Titcomb, quien era el gerente de marketing que cubría los mercados científicos y de investigación. Titcomb tenía muchas ganas de vender algunos KI10 a los sitios que estaban ejecutando TENEX.

El resultado de esto fue que fui a trabajar para DEC, como contratista. Contraté a DEC por un tiempo fijo (3 meses) y precio fijo para hacer que TENEX funcionara en el KI10[6]. Claramente, los procesos internos de DEC no estaban a punto de crear un proyecto real en torno a TENEX en este punto, pero un contrato de 3 meses de un solo hombre debe haber parecido un riesgo aceptablemente pequeño.

También recibí una oferta para unirme a DEC como empleado regular después de que terminara el contrato. Sin embargo, inicialmente no se pretendía que este trabajo permanente tuviera una participación continua con TENEX. De hecho, fue para trabajar en el grupo TOPS-10.

El contrato conmigo para KI-TENEX se negoció a través del departamento de ingeniería de software de DEC, dirigido en ese momento por Dave Stone, a quien conocía anteriormente. Durante la última parte de las discusiones del contrato, Dave le pidió a alguien más que consultara sobre los aspectos técnicos de mi método propuesto para manejar los diferentes hardware de paginación en el KI10. Resultó ser Peter Hurley, que en ese momento realizaba proyectos especiales de ingeniería de software como parte del grupo de marketing encabezado por Allan Titcomb, y marca la primera participación prolongada de Peter en TOPS-20.

Entonces, a principios de octubre de 1972, me despedí de BBN y me instalé en una oficina en 3-5 (edificio 3, piso 5) en los edificios originales de DEC en Maynard. Siendo en ese momento todavía bastante joven en perspectiva, la idea de un contrato de un solo hombre, de precio fijo y de 3 meses para migrar un sistema operativo base a un nuevo procesador en una familia no me pareció particularmente aterrador.

Como parte de mi despedida de BBN, un par de compañeros de trabajo que habían trabajado anteriormente en DEC me dieron algunos regalos de despedida que, me aseguraron, serían muy útiles en DEC: un matamoscas y una lata de repelente de insectos. Las instalaciones de DEC en ese momento carecían de la uniformidad aséptica que generalmente se espera de las oficinas y laboratorios de alta tecnología. Debido a su antigüedad, historia y proximidad a un estanque de molino y un arroyo, los edificios estaban bien provistos de varios insectos y arañas, y mis amigos de BBN querían asegurarse de que yo supiera en lo que me estaba metiendo. En última instancia, pasé solo un poco más de un año trabajando en los edificios de la fábrica de Maynard, pero hubo numerosas ocasiones a última hora de la tarde en las que la posibilidad de una mayor concentración en el código era nula y me encontré observando detenidamente cómo una araña particularmente habilidosa tejía un red circular casi perfecta entre las publicaciones de mi partición.

Algoritmo de paginación en software

La técnica que se me ocurrió no implicaba alteraciones importantes en la estructura de paginación de TENEX para tratar con el buscapersonas más simple del KI10. Más bien, la idea era simular la mayor parte de la lógica del buscapersonas BBN en el software y usar una tabla de páginas KI10 solo como caché de software o búfer de traducción para las asignaciones de páginas virtuales a físicas actuales. Lógicamente, esto era muy parecido al diseño utilizado en muchos procesadores posteriores donde la lógica se realizaría en microcódigo y el almacenamiento en RAM.

La implementación del código PDP-10 para simular el buscapersonas BBN no fue una tarea grande ni difícil y tomó probablemente menos de la mitad del tiempo del proyecto. Además de la paginación, era necesario escribir controladores para el archivo y los dispositivos de intercambio que luego enviaba DEC, ninguno de los cuales se había utilizado en BBN. Sin embargo, la comprobación de TENEX en el KI10 encontró un error lógico nuevo y bastante oscuro en el hardware de paginación.

Mucho antes del final del período de contrato de tres meses, TENEX estaba funcionando lo suficientemente bien en el KI10 para soportar la construcción de nuevas versiones de sí mismo. Durante este período, el sistema estuvo en funcionamiento durante varias horas todos los días en una máquina en el sótano de la planta, y algunos empleados curiosos vinieron a probarlo. Una de esas personas fue Dave Braithwaite, entonces en el grupo de referencia -10, quien trajo varios puntos de referencia y pruebas para probar.

El contrato finalizó nominalmente (con éxito) cuando entregué un conjunto oficial de cintas que contenían las fuentes TENEX y un sistema de arranque en el momento estipulado en el contrato. Sin embargo, esto fue algo académico en ese momento, ya que de ninguna manera fue el final de la actividad relacionada con TENEX en DEC.

Procesador KL10 en desarrollo

Durante el tiempo que trabajé en KI-TENEX, el procesador KL10 estaba en desarrollo activo en ingeniería de hardware. La "L" en KL10 originalmente tenía la intención de significar "bajo costo", ya que el KI10 se percibía como algo costoso. Sin embargo, la tecnología brindaba oportunidades

para dar un salto significativo en el rendimiento y, en última instancia, esa sería la característica más destacada del KL10. Los gerentes de línea de productos estaban viendo oportunidades para crecer en la gama alta, por lo que se preparó el escenario para considerar algunos cambios importantes en las capacidades.

Casi al mismo tiempo, el proyecto "Super Foonly" estaba en progreso en el laboratorio de IA en Stanford. Los diseñadores de Foonly estaban proyectando un factor de aceleración de 10 sobre el KA10, y se invitó a los ingenieros de DEC a revisar estos planes. Ciertas ideas, como la división funcional de E-box y M-box, fueron fundamentales en el diseño definitivo del KL10.

Stanford no era el único otro lugar donde se diseñaba y construía una máquina con arquitectura PDP-10. El recién inaugurado Xerox PARC (Centro de Investigación de Palo Alto) incluía a varias personas de BBN y otros sitios familiarizados con TENEX, y estaban muy interesados en continuar con su trabajo en ese sistema. Dado que Xerox estaba entonces en el negocio de las computadoras, PARC no pudo comprar una PDP-10 de DEC. En consecuencia, construyeron el suyo propio. Era una máquina de un solo procesador compatible con la paginación KA10 y BBN y fue diseñada y construida en un tiempo relativamente corto. Funcionó con TENEX durante varios años.

IBM hace que la memoria virtual sea legítima

Muy posiblemente, el evento fortuito final involucrado en la decisión de DEC de tomar TENEX como base para un producto DEC no ocurrió en DEC sino en IBM. Fue durante este período en la última parte de 1972 que IBM anunció los sistemas de "memoria virtual" para la familia 360. Si bien esto no fue del todo inesperado, proporcionó una gran oportunidad de "legitimidad" para el concepto de memoria virtual en los productos de sistemas informáticos. Yo (y otros defensores de TENEX) habíamos estado promoviendo activamente las capacidades de memoria virtual de TENEX, pero fue necesario el anuncio de IBM para demostrar que tales capacidades podrían ser un factor importante en los grandes mercados de sistemas. Esto es bastante irónico ya que las arquitecturas de administración de memoria en TENEX/TOPS-20 y los sistemas de IBM son bastante diferentes.

Nace DECSYSTEM-20

Pronto, se llevaron a cabo discusiones en torno a la idea de que el KL10 sería, no solo una nueva CPU para el DECsystem-10 existente, sino la piedra angular de una nueva familia de productos de VM que incluye arquitecturas de hardware y software. Las salidas de la arquitectura de hardware debían incluir la memoria principal interna (a diferencia de la memoria de bus externa como en KA y KI), la interconexión Massbus para cintas y discos, y el uso de un PDP 11/40 como consola del sistema. El software daría un paso comparable con las capacidades de TENEX: memoria virtual, estructura de procesos, lenguaje de comandos fácil de usar.

Aunque solo fui parte de algunas de esas discusiones, todavía miro hacia atrás con asombro por la velocidad y la confianza con la que se tomó la decisión de emprender un cambio tan importante. Las personas clave incluían a Allan Titcomb, que había iniciado el proyecto KI-TENEX, Fred Wilhelm, director de ingeniería del proyecto KL10, Bill Kiesewetter, director de marketing de la línea de productos DECsystem-10, y John Leng, director de la línea de productos. No convocamos

grupos de trabajo ni comités de estudio ni divagamos sobre la idea durante uno o dos años. Nos reunimos, consideramos los problemas y decidimos.

Por lo tanto, al final del contrato KI-TENEX de tres meses, recibí una segunda oferta de DEC: unirme al grupo KL10 como líder de proyecto para un nuevo sistema operativo para KL10 basado en TENEX. Cuando comencé a trabajar en DEC como empleado el 2 de enero de 1973, se había contratado a un ingeniero adicional para formar el núcleo de un grupo de desarrollo: Peter Hurley. Los dos establecimos oficinas en el quinto piso del edificio 5 de la planta de Maynard, en un grupo con los ingenieros de hardware, la gente de administración y marketing de la línea de productos, y el vicepresidente Win Hindle.

Comienza el desarrollo de TOPS-20

Hubo una serie de áreas en las que sentimos que era necesario trabajar en TENEX para que sea adecuado para su lanzamiento como producto. Algunas de estas eran áreas en las que TOPS-10 había recibido un trabajo significativo durante los años anteriores, y otras involucraban la mantenibilidad general, el soporte de la confiabilidad y el servicio del hardware y la solidez general. TENEX no tenía, en ese momento, características tales como operación por lotes o estructuras de disco montables.

TOPS-10 Compatibilidad y Coexistencia

También planeamos aumentar significativamente el soporte para la "compatibilidad con TOPS-10": la capacidad del sistema para admitir programas y medios TOPS-10 existentes. Este soporte tomó varias formas: la interfaz de llamada de servicio del sistema operativo, el lenguaje de comandos, los formatos de estructura de disco y otras interfaces externas. TENEX había implementado disposiciones solo para la interfaz de llamadas del sistema operativo, y eso estaba limitado por la falta de conocimiento en BBN sobre los detalles y aspectos no documentados de las llamadas TOPS-10. Además, la compatibilidad de la interfaz de llamadas era tan completa como se necesitaba para ejecutar el conjunto particular de software en capas TOPS-10 que BBN necesitaba en ese momento.

¿Un sistema operativo o dos para máquinas de 36 bits?

Una de las razones para buscar las funciones de compatibilidad con TOPS-10 fue la posibilidad de volver a converger en un sistema operativo para todas las máquinas de 36 bits. Sin embargo, esa fue y siguió siendo una idea controvertida, y el destino de estas características es en gran medida el resultado de la continua indecisión sobre cuánto esfuerzo se debe invertir en herramientas para ayudar a los usuarios a pasar de TOPS-10 a TOPS-20.

A veces, era claramente la intención de que TOPS-20 eventualmente se convirtiera en el único sistema operativo de 36 bits. Parecía tener la capacidad de admitir cualquier nivel deseado de compatibilidad con TOPS-10 y ofrecía características que muchos clientes de TOPS-10 habían estado solicitando pero que parecía poco probable que se pudieran realizar en TOPS-10. Sin embargo, la mayoría de las veces, TOPS-10 (y sus mejoras) parecían ser necesarios a corto plazo para mantener una parte del negocio de 36 bits, por lo que la línea de productos en su conjunto

nunca pudo lograr una resolución suficiente. para mover el negocio de nuevo a un sistema operativo.

En los inicios del -20, la línea de productos hizo algunas propuestas a los clientes del -10 sobre la migración al -20. Las ayudas de migración no estaban en su lugar, TOPS-20 no tenía todas las capacidades necesarias y los clientes de -10 no estaban dispuestos a aceptar los límites en las configuraciones que se aplicaban al -20. Por lo tanto, los clientes reaccionaron bastante negativamente y asustaron a la línea de productos para que no hiciera nada que sugiriera una estrategia de migración.

En última instancia, ambos sistemas operativos continuaron desarrollándose y distribuyéndose hasta que la corporación tomó la decisión mucho más importante de no ofrecer nuevos productos de hardware de 36 bits y reducir todo el desarrollo de software de 36 bits. Por estas y otras razones, reemplazar TOPS-10 no fue una prioridad durante la vida del DECSYSTEM-20 y, por lo tanto, las capacidades como el lenguaje de comando TOPS-10 no se consideraron importantes. Los subsiguientes altibajos de interés en volver a converger en un sistema operativo se describen más adelante.

Interfaz de llamada del sistema operativo

Con la estructura de TENEX, había sido posible implementar la compatibilidad de la interfaz de llamadas TOPS-10 en una forma relativamente modular. Esta pieza de software se llamó PA1050, en honor al sistema DEC 10/50, que era el mejor de la línea -10 en ese momento. Para poder admitir las interfaces de llamada del sistema operativo antiguo (TOPS-10) y nuevo (TENEX), decidimos utilizar un mecanismo de llamada nativo completamente diferente al utilizado por TOPS-10. TOPS-10 usó Monitor UUO, un grupo de instrucciones que transfirieron el control al kernel a través de un vector de transferencia. TENEX tomó un código de operación previamente no asignado (104 octal) y le dio el mnemotécnico JSYS para Jump to SYStem. (El sistema Berekely 940 había utilizado un BRS, Branch to System).

Dentro de TENEX, la ejecución de cualquiera de las llamadas de servicio TOPS-10 se dirigió a una pequeña parte del código del kernel que invocaría PA1050. En la primera llamada de este tipo, asignaría la imagen PA1050 a un área alta de la memoria virtual del usuario y le transferiría el control. En llamadas posteriores, se limitaría a efectuar la transferencia. Por lo tanto, solo aquellos procesos que realmente hicieron llamadas TOPS-10 obtendrían el código de compatibilidad, y cada uno de esos procesos tenía su propia encarnación. La implementación de la memoria virtual de TENEX significó que el código o los datos se podían colocar en cualquier lugar y no hubo penalización por el uso escaso del espacio de direcciones. En ese momento, era bastante seguro usurpar un rango de direcciones altas en un programa TOPS-10, ya que usar tales direcciones en un sistema TOPS-10 real (no VM) habría requerido los 256 Kpalabras completos de memoria o más. correr. Más tarde, a medida que los programas TOPS-10 se hicieron más grandes, tuvimos problemas para comprimir el programa, PA1050 y DDT (que también se asignó a un rango alto de direcciones) todos juntos.

PA1050, entonces, era simplemente un código de modo de usuario sin privilegios. Interpretó los argumentos de la llamada TOPS-10 que había sido interceptada y realizó la acción solicitada,

utilizando llamadas JSYS cuando fue necesario. Mantuvo su propia base de datos del estado del trabajo TOPS-10 en la memoria del proceso local.

Lenguaje de comandos

La idea de lenguajes de comando alternativos fue una de las características atractivas de TENEX. El lenguaje de comandos TENEX (el "EXEC") era simplemente un programa en modo de usuario que operaba dentro de un proceso y realizaba cualquier acción del sistema a través de llamadas JSYS definidas. Por lo tanto, cualquier usuario podría escribir y ejecutar de la misma manera un programa similar para realizar cualquier otra interfaz de usuario deseada sin afectar a otros en el mismo sistema. La gente de marketing tenía visiones de ofrecer interfaces que fueran compatibles con varios sistemas de la competencia para alejar a los clientes de esos sistemas. También imaginamos escribir una interfaz de usuario que se parecería al lenguaje de comandos TOPS-10 para facilitar a los usuarios el cambio de TOPS-10.

De hecho, dicho programa fue escrito durante el primer año de desarrollo en DEC por el tercer miembro del grupo del nuevo sistema operativo, Arnold Miller. Esta interfaz funcionó lo suficientemente bien como para admitir la ejecución de archivos por lotes TOPS-10, así como la mayoría de las actividades normales del usuario. Sin embargo, nunca se envió como parte de TOPS-20, en gran parte porque los usuarios que llegaron a TOPS-20 desde TOPS-10 o cualquier otro sistema rápidamente comenzaron a usar y preferir el lenguaje de comandos TOPS-20. Incluso para los usuarios reacios a probar algo diferente, las ventajas del TOPS-20 Exec eran tan evidentes que pocos querían seguir trabajando en su antiguo entorno. Del mismo modo, nunca se escribieron otras interfaces para imitar los sistemas de la competencia.

El interés en TOPS-10 Exec se reavivó en una o dos ocasiones posteriores cuando se consideró la posibilidad de migrar los usuarios de TOPS-10 a TOPS-20, pero tales pensamientos nunca duraron lo suficiente como para dar como resultado un producto.

Compatibilidad de estructura de disco

Otra característica de compatibilidad desarrollada durante el desarrollo temprano de TOPS-20 pero que nunca se envió fue la E/S de estructura de disco TOPS-10. Esto permitió montar una estructura de disco TOPS-10 en un sistema TOPS-20 y ser leída y escrita por un programa que se ejecuta en modo de compatibilidad TOPS-10. Esto se logró tomando una gran parte del código del sistema de archivos real de TOPS-10 y encapsulándolo con una interfaz para PA1050 en un lado y E/S directa a nivel de bloque en el otro. Nuevamente, esta resultó ser una función que se usó brevemente, si es que se usó, por lo que nunca recibió mucho esfuerzo adicional. Cuando se lanzó la siguiente versión del código del sistema de archivos TOPS-10, resultó más difícil de encapsular y ejecutar de esta manera, y la capacidad se abandonó incluso para uso interno.

El juego de los nombres, o "¿En qué estamos trabajando de nuevo?"

Es hora de discutir la historia de los nombres del sistema operativo que llegó a conocerse como TOPS-20. Como parte del desarrollo de este nuevo sistema operativo, DEC compró los derechos comerciales de TENEX de BBN a principios de 1973. Sin embargo, dado que se esperaba un gran

impulso de marketing para el nuevo sistema basado en KL10, sería necesario un nuevo nombre para el sistema operativo. Además, el grupo de desarrollo de DEC pretendía ser independiente del grupo de BBN, que continuaba desarrollando nuevas versiones de TENEX para los usuarios de KA10 ARPA. En los primeros meses, se consideraron varios nombres. Finalmente, el gerente de ingeniería de KL10, Fred Wilhelm, declaró que el nuevo sistema operativo se llamaría VIROS. Parecía perfecto para la época: sugería "virtual", que iba a ser una característica clave del nuevo sistema, y tenía un sonido fuerte y viril. Desafortunadamente, John Leng (y otros) tendían a pronunciarlo como "virus", por lo que probablemente no habría sobrevivido incluso si otros factores no lo hubieran hecho a un lado.

Posteriormente durante ese primer año, se decidió cambiar el nombre para confundir a la competencia. Esto era parte de una política dictada desde "arriba" para evitar que el mercado supiera demasiado sobre desarrollos no anunciados. Sabíamos que prevenir todas las filtraciones era imposible, por lo que la idea era confundir al mundo con información contradictoria. El nuevo nombre elegido para el sistema operativo entonces fue SNARK, de una historia de Lewis Carol sobre la caza del SNARK, donde "lo perseguían con tenedores y esperanza". (Teníamos muchos de ambos.) Ese nombre casi tuvo una vida corta, ya que se declaró además que los nombres de los códigos internos se cambiarían cada seis meses para mantener al enemigo adivinando. Cuando llegó el momento del cambio de los primeros seis meses, el grupo resistió la molestia de otra perturbación más, pero finalmente cedió y presentó KRANS (SNARK escrito al revés) como el nuevo nombre. Desafortunadamente, pronto se descubrió que esta era una palabra que significaba "bóveda funeraria" en sueco. El nombre se cambió rápidamente de nuevo a SNARK, e incluso nuestra gerencia declaró que no tendríamos nada más que ver con la locura del cambio de nombre semestral.

Por lo tanto, el nombre siguió siendo SNARK hasta que llegó el momento de anunciar el producto. Sin embargo, los diversos nombres provisionales no desaparecieron por completo; todos se usaron para otros fines internos, principalmente como nombres para volúmenes de disco en el grupo TOPS-20. De hecho, el paquete SNARK (pronunciado con acento de Boston como "paquete snak") siguió siendo el hogar de las fuentes TOPS-20 durante toda la vida del sistema. SNARK y KRANS son actualmente nodos en la red interna de DEC pero no están en el grupo de ingeniería de 36 bits.

A medida que se acercaba el momento del anuncio del producto, se estaba dando una consideración de muy alto nivel al nombre del producto. Los meros ingenieros no formaron parte de este proceso, por lo que esperamos con anticipación que se transmitiera el nombre nuevo y final del sistema. ¿Se quedarían con SNARK (poco probable)? ¿Volver a VIROS (los nombres "V" se estaban volviendo más populares todo el tiempo)? O crea algo nuevo y dinámico. El resultado fue, por decirlo suavemente, decepcionante. Aunque TOPS-20 como nombre se convirtió en un hecho de la vida a lo largo de los años, originalmente lo vimos como el nombre menos emocionante y menos imaginativo que uno podría haber elegido para un sistema nuevo y ordenado. Era igual que el antiguo sistema, TOPS-10, excepto por un dígito. Casi tan creativo como "Wednesday Night at the Movies" para un nuevo programa de televisión de la misma época. Pero esa fue la historia: el nuevo producto sería el "DECSYSTEM-20" (alguien olvidó poner en minúsculas el "sistema" como en DECsystem-10) y el sistema operativo sería TOPS-20.

Entonces, aunque el nombre TOPS-20 no se eligió hasta justo antes del envío del primer sistema, he

usado TOPS-20 para referirme al sistema operativo durante todo el tiempo que estuvo en desarrollo en DEC, es decir, desde enero de 1973 en adelante. .

El grupo TOPS-20

El grupo TOPS-20 creció a cuatro personas durante su primer año. Además de mí, Peter Hurley y Arnold Miller mencionados anteriormente, Len Bosack se unió al grupo en los primeros meses. Tom Hastings también estuvo involucrado, sirviendo brevemente como supervisor. Aunque comenzó como parte de la ingeniería de hardware de KL10, este nuevo grupo hizo la transición a su lugar más lógico en la ingeniería de software después del primer año.

Varios otros sirvieron como supervisores del grupo entre su formación y el primer barco. En varias ocasiones, el grupo contrató a su nuevo supervisor. Para ahorrar tiempo, se establecería una entrevista con varios miembros del grupo a la vez que dispararían preguntas al desventurado candidato. Si el candidato no podía sobrevivir a esto, claramente no duraría en el funcionamiento diario del grupo. Los gerentes no fueron los únicos candidatos que recibieron un escrutinio práctico. A Judy Hall, el quinto miembro contratado en el grupo, y a otros que se unieron durante ese período, generalmente se les pedía que trajeran muestras del código que habían escrito previamente cuando acudían a las entrevistas.

Mejoras en la arquitectura del DECSYSTEM-20

El DECSYSTEM-20 (y TOPS-20) se envió por primera vez en enero de 1976, solo tres años después de que comenzara el desarrollo interno de DEC. Esto fue, por supuesto, más largo de lo que habíamos anticipado. Se ampliaron los cronogramas tanto del hardware como del software, y se produjeron algunos otros cambios en los planes. Como se mencionó anteriormente, el KL10 ya se había apartado un poco de su concepción original como un reemplazo KI10 de "bajo costo" cuando comenzó el desarrollo del TOPS-20. A medida que avanzaba el desarrollo de TOPS-20, proporcionamos un par de "1-plusses" más a la máquina.

Direccionamiento extendido

Uno de ellos fue el direccionamiento extendido. TOPS-20 no forzó esto como un problema, pero pareció brindar la oportunidad de aprovecharlo eventualmente. Tom Hastings me sugirió por primera vez la posibilidad de ampliar el espacio de direcciones de la arquitectura PDP-10 durante el período de 3 meses en el que estaba trabajando en KI-TENEX. En ese momento, parecía una idea bastante radical que 256 Kpalabras de espacio de direcciones podrían no ser suficientes para la mayoría de las aplicaciones, pero el PDP-11 ya estaba teniendo problemas en ese sentido y parecía prudente pensar en ello para el -10 mientras estábamos construyendo este nuevo procesador.

Había dos problemas principales que resolver al ampliar el espacio de direcciones del PDP-10. Primero, se necesitaría una forma de generar direcciones efectivas de más de 18 bits; en segundo lugar, se necesitaría un medio para mapear este espacio de direcciones virtuales más grande. Para los diseñadores de hardware, el segundo problema parecía más formidable. Planearon usar una RAM de búsqueda de tablas para realizar traducciones de direcciones virtuales a físicas, y un espacio de direcciones más grande parecía requerir una RAM de traducción prohibitivamente

grande. El KI10 había usado una memoria asociativa para este propósito, pero los circuitos integrados disponibles en la tecnología del KL10 no proporcionaban una forma de hacer una memoria asociativa práctica.

Como sucedió, la solución a este problema resultó de un poco de sinergia. Alan Kotok había estado buscando una forma de hacer que la memoria RAM de paginación fuera más pequeña incluso de lo que parecería ser necesario para el espacio de direcciones básico del PDP-10. Con 512 páginas en cada uno de los espacios de direcciones de modo usuario y ejecutivo, serían necesarias 1024 entradas para un esquema de mapeo directo. Alan estaba pensando en usar las mismas entradas 512 tanto para el usuario como para el ejecutivo almacenando el bit de modo de usuario en la entrada y comparándolo en una búsqueda. Si la comparación falla, la entrada se considerará inválida y se volverá a cargar desde la memoria. Por lo tanto, la misma entrada se usaría para un usuario y una dirección de modo ejecutivo dados y funcionaría tan bien como una tabla de tamaño completo siempre que no hubiera referencias demasiado frecuentes a las mismas direcciones de usuario y ejecutivo.

Mientras esto sucedía, me di cuenta de que no había nada único en el bit de modo de usuario en este esquema y que podía extenderse a un número arbitrariamente grande de bits de dirección. Es decir, un número determinado de bits para traducir podría dividirse en dos partes: una parte para indexar en la RAM y la otra para almacenar y comparar. Para fines de direccionamiento extendido, eso significaba que podíamos tener una cantidad de espacios, cada uno tan grande como los 256 Kpalabras originales, todos mapeados en la misma RAM de traducción. Cuanto más pequeña es la RAM, mayor es la posibilidad de conflicto, pero eso podría mitigarse un poco "haciendo" los bits de dirección para que los diferentes espacios de direcciones se compensen de manera diferente en la RAM.

En cuanto a las modificaciones al conjunto de instrucciones básicas, inicialmente decidimos ampliar el espacio de direcciones de 18 a 23 bits. Se eligió este número porque era el número de bits disponibles al combinar los campos Y, I y X (dirección, indirecto e índice) de la instrucción PDP-10. Tuvimos la suerte de que el conjunto de instrucciones del PDP-10 tenía un diseño muy limpio y consistente: cada instrucción calculaba su dirección de operando de la misma manera. Si este algoritmo pudiera modificarse para generar una dirección más grande, el problema se resolvería esencialmente para todas las instrucciones. La PC se podía hacer fácilmente tan grande como fuera necesario en hardware, y los diversos formatos de palabra en los que aparecía generalmente tenían los 5 bits adicionales disponibles.

A pesar de todo eso, cualquier modificación al conjunto de instrucciones para generar direcciones más grandes sería incompatible con el código existente hasta cierto punto. Para evitar la introducción de tales incompatibilidades, el diseño original tenía una forma especial de ejecutar una instrucción cuando se deseaba una dirección de operando extendida. Una instrucción ejecutada normalmente generaría una dirección compatible de 18 bits, pero la misma instrucción ejecutada como operando de una instrucción de "ejecución especial" (SXCT) podría generar una dirección más grande. Hizo esto usando bits adicionales de una palabra indirecta o índice especificado en la instrucción.

Al final resultó que, nuestro diseño de direccionamiento extendido inicial era demasiado

conservador. Incluso antes de que se enviara el primer DECSYSTEM-20, tuvimos la oportunidad de revisar y modificar en gran medida el diseño del direccionamiento extendido en relación con el proyecto UNICORN que se describe más adelante.

Algoritmo de paginación en microcódigo

Otra perturbación del diseño del KL10 fue en el área de paginación. Debido al éxito de ejecutar TENEX en el KI10 utilizando una simulación de software del diseño de paginación de TENEX, inicialmente se asumió que se podía hacer lo mismo en el KL10. Sin embargo, sabíamos que la simulación del software consumía una cierta cantidad de tiempo del procesador porque la aceleración de los programas TENEX que se movían del KA10 al KI10 normalmente no era tanta como se esperaba. Algunos estudios que utilizaron técnicas de muestreo de PC revelaron que el código de simulación de paginación a menudo usaba más del 30% del procesador bajo una carga típica de tiempo compartido.

Esto trajo a colación una vez más el espinoso problema de poner los complejos algoritmos de paginación de TENEX en el hardware, algo que se había resistido tanto para el KA10 como para el KI10. Dedicar tanto hardware a la paginación todavía parecía una mala idea para los ingenieros de hardware, pero afortunadamente había otra posibilidad para el KL10: el microcódigo. Así como la implementación de KI-TENEX había demostrado que el algoritmo de recarga de página podía ejecutarse en software con el resultado entregado al hardware, también podría ejecutarse mediante microcódigo. El conjunto de instrucciones básico se iba a implementar en microcódigo en el KL10 (el primer procesador de arquitectura PDP-10 así diseñado), por lo que, según pensamos, podríamos agregar "solo un poco más de microcódigo" para el algoritmo de paginación.

Fueron necesarios algunos cambios en el diseño de hardware del KL10 tal como existía en ese momento para permitir que el microcódigo obtuviera el control cuando fallaba un intento de traducción de dirección, calcular la traducción y cargarla en el búfer de traducción, y luego continuar con la instrucción. en curso. Eso se solucionó con poco hardware adicional, y parecía que finalmente habíamos logrado el soporte completo de la arquitectura de paginación. Con un poco de miopía típica de esa época, pensamos que habíamos resuelto todos nuestros problemas de rendimiento de paginación, ya que todo lo que se hacía en microcódigo tenía que ser increíblemente rápido. ("Microlocura", como solía llamarla Len Bosack).

Por lo tanto, el KL10 tenía capacidades extendidas de direccionamiento y paginación TOPS-20 cuando se envió por primera vez. Es posible que esas mejoras extendieran un poco el cronograma de desarrollo, aunque argumentamos en el momento en que se consideró cada una de ellas que el trabajo adicional era insignificante en el panorama general. No estoy muy seguro de cómo finalmente conseguimos que se aprobaran estas cosas. Recuerdo que Win Hindle, bastante enojado, prácticamente nos echó de su oficina la primera vez que entramos para hablarle sobre la adición de direccionamiento extendido a la máquina.

Mejoras de software

Mientras estábamos molestando al grupo de hardware con cambios en la paginación y el direccionamiento extendido, también estábamos ocupados con una gran cantidad de proyectos de

software. Un proyecto típico para hacer que el sistema sea más robusto en entornos comerciales fue el de modificar el manejo del sistema de escrituras en disco y tablas de asignación de disco. En TENEX y TOPS-20, esencialmente todas las E/S de disco se inician mediante paginación. Las páginas de archivos se asignan, ya sea en el espacio de direcciones del usuario o del ejecutivo, y se leen o escriben con instrucciones ordinarias. En TENEX, en lo que respecta a DEC, todas las páginas de archivos se trataban por igual, incluidas las de directorios, tablas de páginas y tablas de asignación de discos. Esto funcionó bien siempre que el sistema nunca fallara. Si el sistema se desactivó intencionalmente, su último acto sería escribir todas las páginas modificadas en el disco y, por lo tanto, hacer que el disco sea coherente.

Sin embargo, si el sistema colapsaba (como ocurría ocasionalmente), podría ser fácilmente que se hubiera escrito en el disco un conjunto inconsistente de páginas. Un directorio de archivos puede estar apuntando a un archivo para el que no se ha escrito la tabla de páginas, o pueden estar en uso páginas para las que no se ha actualizado la tabla de asignación. Para encargarse de esto, la práctica en TENEX había sido ejecutar un programa de verificación de consistencia de disco llamado CHECKDSK. En particular, esto marcaría la tabla de asignación para cualquier página que se encuentre en uso para evitar la asignación múltiple. Esto había sido satisfactorio al principio, pero con el aumento en la cantidad de almacenamiento de archivos, había comenzado a llevar mucho tiempo. La solución fue forzar la escritura de las tablas de asignación, las tablas de páginas y los directorios en ciertos puntos y en cierto orden usando técnicas ahora bien conocidas. Es interesante notar que Unix tuvo problemas similares en sus primeras versiones, y se tuvieron que aplicar las mismas técnicas en versiones más recientes.

El servicio COMND

Otro paso interesante fue el desarrollo de COMND JSYS. Esta llamada al sistema proporciona un conjunto de servicios para implementar el estilo de interfaz de comando TOPS-20 (reconocimiento de escape, ayuda con el signo de interrogación, etc., como se explicó anteriormente). Sin embargo, no formaba parte de TENEX ya que procedía de BBN. Más bien, esas características de la interfaz se codificaron individualmente en cada programa que las admitía y, de hecho, solo EXEC las admitía en general. Otros programas generalmente solo admitían el reconocimiento de nombres de archivos, ya que eso lo proporcionaba la llamada al sistema GTJFN. Incluso la edición básica de la línea de comandos (rubout para eliminar un carácter, control-U para borrar la línea, etc.) no estaba centralizada y varios programas la proporcionaban más o menos o no la proporcionaban en absoluto.

La necesidad de una edición de línea de comandos centralizada se abordó anteriormente en el desarrollo de TOPS-20 mediante la adición de TEXTI JSYS (y los subconjuntos relacionados RDTXT y RDLINE). El uso cada vez mayor de terminales de video había hecho que las disposiciones ad hoc de los diversos programas de utilidad fueran particularmente notorias. Con TEXTI, nos comprometimos tanto a centralizar las funciones de edición de líneas como a hacerlas sensibles al tipo de terminal que se utiliza. En ese momento, aprovechar la capacidad de la terminal de video para borrar un carácter o una línea todavía era impresionante. Algunos sistemas habían podido agregar al menos un mínimo de soporte de edición de línea de terminal de video porque solo admitían entrada de línea y el almacenamiento en búfer de línea se realizaba de manera centralizada. Sin embargo, TENEX había sido diseñado con un máximo de flexibilidad de terminal

y, por lo tanto, no tenía una forma general de editar la entrada. El TEXTI JSYS proporcionó muchas opciones para que también pudiera ser utilizado por programas que querían más o menos entradas de línea a la vez.

Sin embargo, la implementación de una biblioteca de entrada de comandos centralizada no fue un elemento que apareció en nuestros horarios. En el último año antes de que se enviara el sistema, habíamos identificado y estábamos trabajando en algunos programas de utilidades que necesitaban mejoras específicas. Uno de ellos era DUMPER (la utilidad de copia de seguridad y restauración de disco a cinta), y tenía una interfaz de comandos particularmente mala. Tenía la tarea de mejorar la interfaz de usuario de DUMPER para lograr un nivel mínimo de aceptabilidad, pero había estado postergando hacer algo al respecto porque sería una tarea bastante rutinaria y aburrida. Mientras tanto, comencé a pensar en cómo este estilo de interfaz de comando podría centralizarse en un conjunto de funciones y tablas.

Debido a la naturaleza altamente interactiva de la interfaz, es necesaria la ejecución parcial del programa durante la entrada del comando. Por ejemplo, si se intenta reconocer el nombre del archivo, los valores predeterminados y cualquier otro contexto pertinente deben estar disponibles para que se pueda determinar el éxito o el fracaso. Sin embargo, el programa no debe realizar ninguna acción irrevocable hasta que se confirme el comando (normalmente con un retorno de carro) para que el usuario pueda borrar todo o parte de él con las teclas de edición de línea. Incluso el TENEX Exec original no entendió esto del todo bien. El usuario podía borrar toda la línea de comando con control-U, pero no podía borrar hacia atrás más allá del campo actual. En otras palabras, el ejecutivo sabía cómo vaciar todo el estado de la línea actual y comenzar de nuevo, y sabía cómo eliminar caracteres del campo actual antes de que se completara, pero no sabía cómo vaciar parte del estado y copia de seguridad en un campo ya completado.

Mi idea para resolver el problema de la copia de seguridad no era hacer una copia de seguridad en absoluto, sino guardar el texto del comando y luego volver a escanearlo desde el principio con la parte no deseada eliminada. Con esa idea y una lista de funciones para manejar los diversos tipos de campos que se producirían en una línea de comando, comencé a escribir una instalación de tipo biblioteca en lugar de solo mejorar puntos para DUMPER. Por supuesto, DUMPER estaba destinado a ser, y se convirtió, en el primer programa en usar la nueva función, pero en el momento del primer lanzamiento de TOPS-20, estas rutinas de análisis de comandos se habían integrado en el monitor como COMND JSYS. Sin embargo, la mayoría de las utilidades no se habían modificado para usarlo y el Exec aún tenía su código de análisis original. La corrección de todo esto se hizo en versiones posteriores.

En última instancia, la mayor parte del desarrollo de TOPS20 realizado para la primera versión se destinó a admitir las nuevas arquitecturas y dispositivos de hardware. Era necesaria una estructura de E/S de almacenamiento masivo completamente nueva para los discos y cintas Massbus, y era necesario un cambio igualmente grande para manejar las comunicaciones de E/S a través del front-end PDP-11 en lugar de a través de escáneres de línea directa.

Lenguaje de implementación

Casi todo el código del sistema para las máquinas con arquitectura de 36 bits se había escrito en

MACRO desde la primera rutina de arranque del PDP-6. Debido a su estructura regular y su potente conjunto de operaciones, el lenguaje ensamblador de 36 bits era razonablemente agradable de escribir y, por lo tanto, existía menos presión para un lenguaje de nivel superior que en la mayoría de las otras máquinas. BLISS estaba ganando adeptos en algunas partes de DEC, pero sus diversas peculiaridades (por ejemplo, la notación de "puntos" y el uso de guiones bajos para el operador de asignación en la versión anterior de -10) generaron una gran resistencia en otros lugares. Por lo tanto, el nuevo código TOPS-20 se escribió en MACRO, como lo había sido TENEX.

Durante el desarrollo de la versión 1 y posteriormente, se agregaron varias funciones a las convenciones de programación MACRO utilizadas en TOPS-20. Estos fueron implementados principalmente por macros, y le dieron ciertas capacidades de lenguaje de nivel superior a MACRO. El primero de estos mecanismos involucrados para representar estructuras de datos en un lugar (es decir, un archivo de "encabezado") de modo que la representación provoque que se genere el código apropiado para las referencias. Esto comenzó como una macro simple para seleccionar la "mitad izquierda" o la "mitad derecha" de una palabra de 36 bits (en lugar de un HLxx o HRxx explícito). Luego vinieron las macros para seleccionar la instrucción de prueba de bits correcta (TLxx, TRxx, TDxx) para la máscara en cuestión. Finalmente, las macros fueron capaces de definir y representar estructuras complejas de tipo registro de varias palabras con campos de varios tamaños.

En segundo lugar, se usaron macros para proporcionar llamadas a procedimientos y almacenamiento automático (pila) usando variables con nombre. El punto de entrada de un procedimiento podría declararse usando nombres simbólicos para los parámetros, y estos nombres se expandirían a la referencia de pila apropiada cuando se usaran en una instrucción. De manera similar, el almacenamiento de pila local podría definirse simbólicamente y los registros locales se guardarían automáticamente. Estas convenciones redujeron en gran medida la aparición de instrucciones PUSH y POP de pila explícitas y los errores que a menudo resultaban de ellas.

Finalmente, se implementaron macros para proporcionar estructuras de control sin el uso explícito de etiquetas. Semánticamente, se parecían a las construcciones IF/THEN o IF/THEN/ELSE del lenguaje típico, y se permitía el anidamiento arbitrario. Como resultado, una página típica de código TOPS-20 que usaba estas convenciones a menudo contenía solo una etiqueta: el punto de entrada del procedimiento.

Todo esto mejoró la programación, pero, por supuesto, no eliminó la dependencia del conjunto de instrucciones del PDP-10. La portabilidad no fue un objetivo que pensáramos preocuparnos hasta que fue demasiado tarde.

El KL10 se convierte en un producto

Las consecuencias del ambicioso conjunto de nuevas arquitecturas las estaban sintiendo tanto los ingenieros de hardware como nosotros en el software. A medida que avanzaban los cronogramas, no pasó mucho tiempo antes de que a alguien se le ocurriera la idea de enviar solo el KL10 como una actualización del procesador para los sistemas DEC-10 existentes. Después de todo, la introducción de nuevos procesadores estaba bien establecida y parecía una fuente de ingresos significativa mucho antes de que el DECSYSTEM-20 completo estuviera listo. Fruto de ello nació el proyecto y producto "1080".

El 1080 era el procesador KL10 con adaptadores para conectar a las memorias externas KI10 y al bus IO. Ni la memoria externa ni el bus de E/S estaban destinados a ser parte de la nueva arquitectura, pero con el interés de generar algunos ingresos del costoso desarrollo del KL10, los adaptadores se agilizaron. Dado que el 1080 se vendería a los clientes existentes como actualizaciones de los sistemas existentes, se entregaría solo con el sistema operativo existente, TOPS-10. No tomó mucho trabajo para que TOPS-10 se ejecutara en el KL10; la máquina incluso tenía un modo que emulaba la paginación KI10 en lugar de la nueva paginación basada en microcódigo diseñada para TOPS-20. Entonces, después de toda la emoción original en torno a la unión de TOPS-20 y el procesador KL10, fue TOPS-10 el que se envió en los primeros KL10 que se vendieron.

El primer barco 1080 en agosto de 1975 fue la ocasión para una gran celebración frente a la planta de Marlboro. Toda la línea de productos de 36 bits se había trasladado a la planta de Marlboro a principios de 1974, lo que significaba que teníamos los grupos de fabricación, ingeniería, marketing y todos los demás en un solo lugar. Más tarde, esta naturaleza autónoma de la línea de 36 bits, junto con el hecho de que continuamos construyendo una arquitectura cada vez más a partir de la corriente principal de la empresa, llevó a algunos a referirse a la línea de productos 10/20 como " La empresa de informática Marlboro".

Para el barco 1080, la mayoría de los grupos de Marlboro se reunieron para pronunciar discursos frente a un gran camión digital con destino al primer cliente 1080. En realidad, el camión estaba vacío en ese momento debido a algunos inconvenientes de última hora, pero sustancialmente el producto estaba listo y comenzó la fiesta.

Anuncio de DECSYSTEM-20 y primer envío

Mientras tanto, el desarrollo continuó en el DEC-20. Con las nuevas arquitecturas y la velocidad de la CPU, el sistema sería mucho más rentable que los sistemas anteriores. Debido a esto, la línea de productos retrasó su anuncio hasta poco antes de que estuviera listo para enviarse. Todo esto tuvo lugar en enero de 1976. La primera media docena de clientes del DEC-20 fueron cuidadosamente seleccionados para lo que realmente sería una prueba de campo del nuevo sistema.

Durante los primeros dos o tres años después de que el DEC-20 comenzó a enviarse, disfrutó de cierto favor en DEC. Sin embargo, incluso en el momento del primer envío, había señales claras de que la corriente principal de los productos DEC estaría en otra parte: a la familia PDP-11 le estaba yendo muy bien y estaba ganando mucho dinero para la corporación, y el diseño ya estaba en marcha para la máquina que luego sería conocida como VAX. Sin embargo, hubo un tiempo en que, en retrospectiva, alguna historia significativa pendía de un hilo.

Cómo el VAX casi tenía 36 bits

En 1975, quedó claro que la arquitectura PDP-11 había alcanzado su punto máximo. En particular, el espacio de direcciones era demasiado limitado para admitir muchas aplicaciones en crecimiento. Se inició un esfuerzo para construir una nueva máquina que se vendería en el extremo superior del rango de precios de PDP-11 y más allá y sería el camino de crecimiento para los usuarios de -11.

Esta nueva máquina, cuyo nombre en código era UNICORNIO, se basaría en la arquitectura de 36 bits dado que ya había un paquete de software disponible para ella. Varios de los ingenieros de más alto nivel en los grupos PDP-11 comenzaron a venir a Marlboro para hablar sobre la construcción de un -10 pequeño, pequeño para nuestra forma de pensar, pero grande en comparación con la línea -11.

Un resultado de este trabajo fue que se revisó el diseño original para el direccionamiento extendido. Con metas más ambiciosas para el negocio y el rendimiento, y una mayor apreciación por el dolor de quedarse sin espacio para direcciones, los ingenieros de PDP-11 insistieron en mejorar el diseño de direcciones extendidas para mejorar la eficacia final. Nos convencieron de que el rendimiento no debe verse comprometido por razones de compatibilidad o conversión como se había contemplado en el diseño original.

Surgió un nuevo diseño de direccionamiento extendido que fue una gran mejora a largo plazo. Sin embargo, era demasiado tarde para implementarlo por completo en el producto KL10 inicial. El nuevo diseño incorporó una dirección de 30 bits en lugar de 23. También eliminó la "ejecución especial" porque eso habría agregado un ciclo adicional a cada referencia extendida y habría afectado el rendimiento de los programas que utilizan en gran medida el direccionamiento extendido. En su lugar, se proporcionó compatibilidad en función del uso del espacio de direcciones. El espacio de direcciones general se dividió en "secciones" de 256 Kpalabras cada una. En la sección 0, denominada "sección conmemorativa de KI", el código funcionaría exactamente como lo había hecho en el KI10, es decir, sin posibilidad de direccionamiento extendido. Sin embargo, si la PC estuviera en una sección distinta de 0, se usaría un algoritmo diferente para el cálculo de la dirección, de modo que se podría generar una dirección extendida (intersección) en cualquier momento. Estas diferencias significaban que el código no era estrictamente compatible, pero el estilo y la sensación general eran muy parecidos. En cualquier cuerpo de código dado, la mayoría de las instrucciones no necesitarían cambiarse para ejecutarse en el entorno extendido, por lo que sentimos que todavía habíamos proporcionado un buen camino para la migración del código existente.

Irónicamente, como ya te habrás dado cuenta, el proyecto UNICORN nunca llegó a buen término. En un tiempo relativamente corto, se llegó a la conclusión de que, incluso si se pudiera construir una máquina de arquitectura barata de 36 bits, no sería "culturalmente compatible" con el software y las aplicaciones del PDP-11 y, por lo tanto, no satisfaría la necesidad. En su lugar, se formó un grupo central para diseñar una arquitectura que sería culturalmente compatible con el PDP-11 pero que eliminaría las limitaciones del espacio de direcciones del PDP-11. Eventualmente, la máquina construida como resultado de este esfuerzo fue nombrada VAX-11, por extensión de dirección virtual del 11. Sin embargo, como sabemos, los diseñadores hicieron mucho más que simplemente extender el espacio de direcciones. Lo que no es muy conocido es que, al menos durante un tiempo, ¡se planeó que el VAX tuviera 36 bits!

Otros efectos del desarrollo de VAX en TOPS-20

El direccionamiento extendido no es la única forma en que VAX impactó la primera versión del -20. Cuando se envió el -20, el diseño ya estaba en marcha para el software y el hardware de VAX. En un esfuerzo por reducir la plétora de diferentes interfaces de comando en los productos DEC, se

formó un comité para especificar un lenguaje de comando estándar DEC, o DCL. En ese momento, los lenguajes de comando del sistema existentes incluían TOPS-10, TOPS-20 y varios sistemas PDP-11: RSX, RSTS, RT-11, IAS y otros. Como parte de la estrategia emergente de "un sistema", este nuevo estándar de lenguaje de comandos tenía la intención de afectar los sistemas existentes siempre que fuera posible y, especialmente, para guiar la implementación de todo el nuevo software VAX.

Hubo varias áreas controvertidas en las que el ejemplo TOPS-20 fue finalmente persuasivo y se utilizó como base para el estándar. Por ejemplo, se adoptó el orden de TOPS-20 de COPY (COPY from to) en lugar del estilo de asignación (COPY to=from) de TOPS-10 y otros. Sin embargo, en un área, TOPS-20 acordó un cambio en respuesta al consenso del comité. Hasta ese momento, TOPS-20 había continuado con la práctica de TENEX de usar punto y coma como puntuación para el número de generación del archivo, p. NAME.EXT;3 para la generación 3. Bastante tarde en el proceso, se dio cuenta de que la generación no requería un carácter de puntuación diferente al de la extensión y que sería más económico usar el mismo, p. NOMBRE.EXT.3. Estuvimos de acuerdo con eso, aunque significaba cambiar algo de código muy cerca de la fecha del lanzamiento 1 de TOPS-20. No obstante, lo hicimos para no tener que enfrentar el problema de hacer un cambio incompatible en una versión posterior.

Desafortunadamente, los desarrolladores del software VMS ya habían comenzado a usar el punto y coma y se resistieron a hacer el cambio. En última instancia, el cambio nunca se realizó y el software VMS se envió con punto y coma. El estándar se modificó para volver al punto y coma, pero TOPS-20 ya se había enviado utilizando el punto. Por lo tanto, se perdió una pequeña oportunidad de compatibilidad con 20/VAX a pesar de nuestros mejores esfuerzos para lograrlo.

TOPS-20 durante sus años de producción

Aunque el fin formal de la familia de productos de 36 bits no se anunció hasta 1983, la vida en la tierra de los 36 bits parecía precaria durante muchos años antes de eso. El KS10 se construyó como una adición de gama baja a la familia y se envió como "2020" aproximadamente un año después de la introducción de la línea 20, pero ese sería en última instancia el último procesador nuevo que se presentaría. Sin embargo, se iniciaron varios otros.

delfín y pececillo

Uno era conocido como Dolphin y sería una máquina de gama alta con un rendimiento mucho mayor que el KL10. Sin embargo, se llevó a cabo como un desarrollo combinado de 36 bits/VAX. Es decir, la mayor parte de la ingeniería estaba destinada a aplicarse por igual a un procesador VAX o de 36 bits, y se suponía que solo se necesitaba un poco de trabajo específico de la arquitectura para producir cada máquina. Esto, junto con otras complejidades de la tecnología y los objetivos de rendimiento, hizo que el proyecto fuera más allá de las capacidades disponibles en ese momento. Los costos aumentaron y el proyecto se atascó. A principios de 1979, la parte de 36 bits del proyecto se canceló y solo la parte VAX, entonces llamada VENUS, continuó y finalmente produjo el VAX 8600.

Otro proyecto cancelado más o menos al mismo tiempo fue un PDP-10 muy pequeño conocido

como Minnow. Este proyecto, que operaba en una pequeña trastienda en el segundo piso de la planta de Marlboro, había construido una placa de prueba que funcionaba lo suficientemente bien como para demostrar algunos programas de modo ejecutivo. Consistía en tres placas de PC grandes: una placa de procesador que incluía 8 UARTS para líneas de comunicación, una placa de memoria y una placa controladora de disco. Estos podrían colocarse sobre un escritorio, pero se concibieron para residir en un gabinete tipo escritorio con un disco fijo y otro extraíble. Si esta máquina no hubiera sido cancelada, creo que habría sido una entrada significativa en lo que luego se convirtió en el mercado de las estaciones de trabajo. No era una estación de trabajo completa, ya que no habíamos anticipado el valor de tener un sistema de visualización integrado, pero habría sido una máquina "personal" y un producto con costo/rendimiento varios años por delante del mercado.

Sucumbió en parte a la determinación corporativa de adherirse a la estrategia de arquitectura única de VAX. Si bien el producto Minnow parecía atractivo, se decidió que enviaría un mensaje equivocado al introducir una máquina que no fuera VAX en ese rango de precios. Además, Minnow también tenía un apoyo limitado en los grupos 10/20. Muchas personas consideraban que las máquinas de 36 bits vivían únicamente en salas de ordenadores y estaban rodeadas de grandes discos y unidades de cinta, y no podían entender cómo una pequeña en una oficina podría ser útil. Esto parecía ser parte de un conservadurismo mayor que impidió o desanimó intentos significativos de sostener y expandir el negocio de 36 bits.

La estrategia de "salir del negocio"

Por lo tanto, en 1979, la línea de 36 bits se había cancelado esencialmente de nuevo [consulte la sección 1.2]. Sin embargo, la corporación no estaba del todo lista para declarar esto al mundo. A los VAX les estaba yendo bien, pero no estaban listos para hacerse cargo de todo el negocio de alta gama. A raíz de la cancelación de Dolphin, se decidió construir una última máquina con arquitectura PDP-10, que se llamaría "2080". Este proyecto estaría separado de cualquier desarrollo de VAX. Sería simple, de bajo costo, predecible y limitado a producir un sucesor rentable del KL10. Estos objetivos limitados llevaron a una mentalidad de quiebra entre los grupos de 36 bits. Los planes ambiciosos oa largo plazo se archivaron y los proyectos se reenfocaron en resultados a corto plazo. El único propósito oficial para construir el 2080 fue "apoyar a los clientes actuales", es decir, no intentar hacer crecer el negocio.

Un área afectada por esto fue TOPS-10 y las perspectivas de una estrategia de un solo sistema. Mientras continuaba el desarrollo de Dolphin, se invirtió algo de esfuerzo en considerar lo que se necesitaría para hacer que TOPS-20 fuera totalmente capaz de admitir a todos los clientes de 36 bits. Se habían realizado estudios de rendimiento para encontrar áreas en las que el rendimiento de TOPS-20 fuera inferior al de TOPS-10. Se consideraron problemas de redes y multiprocesamiento. Sin embargo, con la cancelación de Dolphin y la estrategia de cierre del negocio, claramente no tenía sentido trabajar para mover los clientes TOPS-10 a TOPS-20, por lo que esos objetivos se descartaron una vez más. Cada grupo de sistemas continuó con el desarrollo que le pareció pertinente, en muchos casos realizando esencialmente proyectos duplicados de soporte de hardware.

Otra víctima probable de horizontes limitados en tierra de 36 bits fue TOPS-20 SMP. La arquitectura de TOPS-20 se adaptaba bien a SMP porque se consideraron las necesidades de ese estilo de operación cuando se diseñó TENEX. El sistema no solo habría soportado fácilmente el

tiempo compartido de SMP, sino que la estructura del proceso también habría permitido que los programas de usuario iniciaran fácilmente actividades paralelas dentro de un solo trabajo.

El producto TOPS-20 inicialmente no era SMP porque el hardware DECSYSTEM-20 era inherentemente de una sola CPU. Con memorias externas, como las que se usaban con las configuraciones TOPS-10, se podría admitir SMP. De hecho, TOPS-10 SMP es uno de los aspectos más exitosos y ampliamente estudiados de TOPS-10. Sin embargo, como decisión de producto, el DECSYSTEM-20 NO se configuraría con memoria externa y, por lo tanto, TOPS-20 no se podría ejecutar en una configuración de hardware compatible con SMP. El 2020 también fue de una sola CPU. Se planeó que Dolphin fuera una máquina multiprocesador, y SMP se incluyó en los planes iniciales de TOPS-20 en ese momento, pero la cancelación de Dolphin eliminó esa posibilidad para una plataforma de hardware.

Curiosamente, TENEX se modificó para admitir SMP, pero no en DEC o BBN. Se ensambló una configuración KI10 de dos procesadores en el Centro Médico de la Universidad de Stanford, y Rainer Schultz de Stanford modificó KI-TENEX para que funcionara como un sistema operativo SMP completo. Este sistema estuvo operativo alrededor de 1977 y se llevaron a cabo algunas conversaciones con DEC sobre la adición de las modificaciones SMP a TOPS-20. Sin embargo, nunca salió nada de esto, ya que el producto de hardware -20 definido nunca incluyó una posible configuración SMP.

Planes ambiciosos para Júpiter

A medida que el proyecto 2080 continuaba, siguió la larga tradición de 36 bits de recoger algunos "1-plusses" aquí y allá. En poco tiempo, había desarrollado algunas proyecciones de rendimiento muy ambiciosas debido en gran parte al uso planificado de técnicas de canalización. El conjunto de instrucciones PDP-10, al ser relativamente simple (aunque no tan simple como un conjunto de instrucciones RISC), parecía prestarse a una gran cantidad de canalización, particularmente si los compiladores podían generar secuencias de código óptimas. Empezó a parecer que la experiencia KL10 podría repetirse y darle a la arquitectura de 36 bits una nueva oportunidad de vida.

Otro cambio al plan original fue la inclusión de la estrategia de interconexión DEC que entonces surgía. Esto requería que los dispositivos de almacenamiento masivo se conectaran a través de los controladores CI y HSC, y que otros periféricos se conectaran a través de Ethernet. Aproximadamente dos años después de iniciado el proyecto, el sistema se redefinió para usar estas nuevas interconexiones exclusivamente, descartando el soporte para massbus, Unibus y otras interconexiones existentes. El proyecto pasó a llamarse JUPITER, la línea de productos LCG renació y los pensamientos de cerrar el negocio habían desaparecido en gran medida del grupo.

Se realizaron importantes esfuerzos de software en relación con esto. Más allá del soporte básico de hardware, se nos ocurrió el concepto de Common File System como el medio para soportar sistemas con múltiples CPU. Se estaba realizando un trabajo similar en el grupo VAX/VMS y llegó a ser conocido como Clusters. En última instancia, tanto VMS como TOPS-20 enviaron este tipo de capacidad. Las implementaciones fueron diferentes y reflejaron las diferencias en los dos sistemas. VMS usó un administrador de bloqueo de clúster como el método básico para regular el acceso simultáneo a los archivos, mientras que TOPS-20 incrustó el bloqueo en las rutinas primitivas

básicas de apertura de archivos. La implementación de TOPS-20 también continuó utilizando el mapeo de memoria virtual como paradigma para hacer referencia a los archivos, y es posiblemente el único sistema que implementó el acceso distribuido a la memoria virtual compartida.

Una vez más, se generaron algunas ideas en torno a la posibilidad de pasar a una estrategia de sistema operativo único para máquinas de 36 bits. Durante un breve período, se imaginó un sistema que combinaba de alguna manera todas las bondades de TOPS-10 y TOPS-20. Esto se estaba llamando TOPS-36 para no hacer pensar a los partidarios de TOPS-10 o TOPS-20 que era solo el otro. Es imposible decir cómo podría haber resultado esto, ya que nunca se hizo ninguna planificación o diseño más allá de los escenarios generales.

La mayoría de estos ambiciosos planes se redujeron o abandonaron porque Júpiter no repitió la historia de KL10; más bien era más como la historia de los delfines. Nunca se puede saber con certeza si el proyecto fue "más de 1" demasiado o si se cometieron otros errores evitables. Sin embargo, después de haber recibido una gran extensión de tiempo y una infusión adicional de fondos de desarrollo, el proyecto descubrió que iba a necesitar mucho más de ambos. Además, algunos análisis retrasados durante mucho tiempo del rendimiento de ciertas instrucciones comunes de 36 bits llevaron a la conclusión de que el rendimiento general no estaría a la altura de las expectativas. De hecho, difícilmente superaría al KL10 en algunos casos. Probablemente habría sido posible solucionar esos problemas, pero el proyecto ya estaba muy por encima de su cabeza y parecía atascado en replanes semanales e indecisión.

Conclusión

La temprana popularidad de TENEX en ARPANET fue sin duda una clave para su posterior aceptación, transformación y soporte como TOPS-20 por parte de DEC. Esa popularidad, a su vez, parece haberse basado no solo en sus características técnicas, sino también en el hecho de que se desarrolló dentro de esa comunidad y respondió a ella. Cuando TOPS-20 se convirtió en un producto DEC, se convirtió en parte de un mercado mucho más grande y, por lo tanto, menos receptivo a cualquier segmento particular del mercado.

Además de eso, a medida que aumentó el interés en otras arquitecturas informáticas a finales de los años setenta, muchos sitios de investigación llegaron a la conclusión de que no querían depender de ningún proveedor de hardware o software y, en cambio, querían sistemas "abiertos". que sería susceptible de modificación y evolución local. Esto, por supuesto, condujo a un rápido aumento en el uso y la popularidad de Unix. El hecho de que Unix se implementara en un lenguaje razonablemente portátil (al menos en comparación con MACRO de 36 bits) también alentó su difusión a nuevas arquitecturas de chips y más allá. Si hubiera podido hacer algo diferente en la historia de TENEX y TOPS-20, habría sido codificarlo en un lenguaje de nivel superior. Con eso, es probable que el sistema, o al menos una gran parte de él, se haya extendido a otras arquitecturas y finalmente haya sobrevivido a la desaparición de la arquitectura de 36 bits.

Gran parte del éxito de TOPS-20 y otro software de 36 bits es el resultado de las cualidades que surgieron de los entornos interactivos en los que se creó y utilizó. En los últimos 10 años, la fuerza motriz de los entornos interactivos ha sido acercar el poder de la computadora al usuario a través de computadoras personales y estaciones de trabajo. Para que TOPS-20 se mantuviera a la vanguardia

del desarrollo y uso interactivo, se tendrían que haber construido plataformas de hardware de 36 bits que fueran lo suficientemente pequeñas y baratas para ir a la oficina, o el software tendría que haber sido lo suficientemente portátil para pasar a otras plataformas similares.

Aunque este libro trata sobre la arquitectura de 36 bits de DEC, ahora está claro que las arquitecturas de CPU de hardware están perdiendo importancia en la configuración del software. Durante mucho tiempo, las arquitecturas de conjuntos de instrucciones impulsaron la creación de nuevo software. Se introduciría una nueva arquitectura y se escribirían nuevos sistemas de software para ella. La arquitectura de 36 bits era grande en comparación con la mayoría de los otros sistemas que permitían el uso interactivo. También tenía un costo más bajo que la mayoría de los otros sistemas de ese tamaño. Estos factores fueron importantes en la creación del tipo de software por el que se conoce la arquitectura.

Ahora, las nuevas arquitecturas están apareciendo con una frecuencia cada vez mayor, pero simplemente se "deslizan" debajo del software. Los sistemas de software son demasiado grandes para reescribirlos cada vez, y un sistema que no puede adaptarse a las nuevas arquitecturas eventualmente sufrirá una disminución del interés y la pérdida de plataformas de hardware competitivas. TOPS-20 no pasó esa prueba, aunque aportó una serie de ideas a la tecnología de los sistemas interactivos. Hasta dónde se difundieron finalmente estas ideas es una historia que aún no se ha contado.

REFERENCIAS

1. "TENEX, un Sistema de Tiempo Compartido Paginado para el PDP-10". Daniel G. Bobrow, et al. CACM, Vol 15 No 3, marzo de 1972.
2. "Estructura de un sistema LISP que utiliza almacenamiento de dos niveles". Daniel G. Bobrow y Daniel L. Murphy. CACM, Vol 15 No 3, marzo de 1967.
3. "Una nota sobre la eficiencia de un cálculo LISP en una máquina paginada". Daniel G. Bobrow y Daniel L. Murphy. CACM, Vol 11 No 8, agosto de 1968.
4. "El modelo de conjunto de trabajo para el comportamiento del programa". Peter J. Denning. CACM, Vol 11 No 5, mayo de 1968.
5. "Organización y Gestión del Almacenamiento en TENEX". Daniel L. Murphy. Actas AFIPS, 1972 FJCC.
6. "Implementación de TENEX en el KI10". Daniel L. Murphy. Sesión del Panel TENEX, NCC 1974.